Volume 5, Issue 5, May 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Parallel Algorithm for Hermite Interpolation on the Star Graph

B N B Ray
Dept of Computer Science, Utkal University,
Odisha, India

Abstract— In this paper we have developed a parallel algorithm for N points Hermite Interpolation on star graph. The algorithm has three phases: initialization, main and final . The algorithm has N/2 data communications, $2N+6$ ***multiplications,*** $2N+1$ ***subtractions,*** N ***additions and*** N ***divisions.***

Keywords:-Star graph; Hermite Interpolation; data communications ;parallel algorithm; network;

I. INTRODUCTION

Lagrange interpolation on the *n*-star graph. The main phase of the algorithm consists of *n*!/2 steps and each step consists of four multiplications, four subtractions and one communication operation, and an additional step including one division and one multiplication.

The star graph, a member of the class of Caley graphs, has been proposed in [6] as an attractive alternative to the hypercube topology for interconnecting processors in parallel computers. It has been extensively studied in different aspects and many algorithms have been designed for it including sorting [11] , load balancing [13], computational geometry algorithms [3] , Fourier transform [8] and communication algorithm [2].

A. *Parrallel algorithm on star graph*

In this section we propose a parallel algorithm for computing Hermite interpolation on the star graph . The algorithm combines several communication techniques for computing an N-point Hermite interpolation on an N-node star graph. Such a combined method can be adapted for computing any function $f(x)$, with the property that deriving the value of the function for a given input x needs all other known dataset ($(x_i, y_i = f(x_i), y_i' = f'(x_i))$. Examples of such functions exist in numerical analysis and image processing domains. The method can also be applied for any Hamiltonian network, but the performance will vary from one network to other depending on communication cabilities of the host network. A network is Hamiltonian if a cycle can be embedded in the network, which includes all of its nodes [5,13]. The algorithm relies on all-to-all broadcast communication at some stage during computation, as will be discussed later. This is achieved by using a gossiping algorithm on a Hamiltonian ring embedded in the host star network.

II. PRELILMINARIES

This section gives some preliminaries which are necessary to develop and describe our parallel algorithm later.

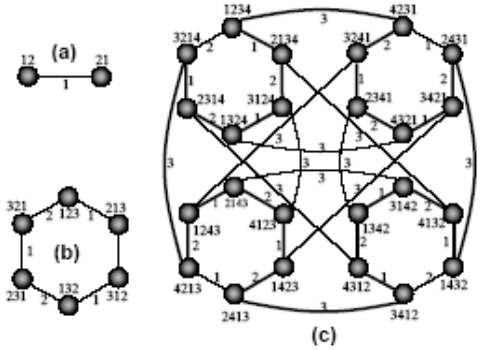

Fig.1- The star graph: (a) 2-star, S2. (b) 3-star, S3. (c) 4-start, S4.

A. *The Star Graph.*

The n-star graph, denoted by S_n, has *n!* vertices (or nodes) corresponding to the *n!* permutations of *n* distinct symbols. A vertex corresponding to permutation $a_1a_2...a_{i-1}a_ia_{i+1}...a_n$ is connected to those vertices corresponding to permutations $a_ia_2...a_{i-1}a_1a_{i+1}...a_n$ for $2 \le i \le n$, (that is, those permutations that result from interchanging the first

symbol in the permutation $a_1a_2...a_{i-1}a_ia_{i+1}...a_n$ with any of the remaining *n*-1 symbols). The edge connecting the vertex associated with the permutation resulting from the interchange between their first and the *i*th symbols is called the *i*th connection. Thus, we can define a function, Γ, to give the permutation address of the node connected to a given node, $a_1a_2...a_{i-1}a_ia_{i+1}...a_n$,, via the *i*th connection as $\Gamma_i(a_1a_2...a_n) = a_ia_2...a_{i-1}a_1a_{i+1}...a_n$. In this way, every vertex is an endpoint of *n-1* edges, corresponding to the *n-1* symbols that can be interchanged with the symbol in the first position of the associated permutation. This is shown in Figure 1 for three different size of star, the 2-star S_2, 3-star S_3, and 4-star S_4.The degree and the diameter of the n-star is $O(n) = O(\log n!/\log n)$ while it is $O(\log n!) = O(n \log n)$ for the equivalent hypercube (with the same number of nodes *N= n!*)[1].

B. Processor Ordering

We need an ordering for the nodes of the *n*-star and a function that maps this ordering to the positive integers in order to develop our algorithm. The processor ordering defined in [1] is used where an ordering, $\prec$, on the nodes of *n*-star is defined as follows. We say $a_1a_2...a_i...a_n \prec b_1b_2...b_i...b_n$ if there exists an i, $1 \le i \le n$, such that $a_j = b_j$ for j>i and $a_i > b_i$. Let us now describe a function that maps the permutations ordered according to $\prec$, into the first *n!* integers, 1,2… , and *n*!. For each permutation, $a_1a_2...a_i...a_n$, we define π_i, for each a_i, with $2 \le i \le n$, to be

$$\pi_i = \left|a_i - i - \sum_{j=i+1}^{n} (a_i > a_j)\right|(i-1)! \quad \text{(2.1) where } (a_i > a_j) = \begin{cases} 1 & \text{if } a_i > a_j \\ 0 & \text{otherwise} \end{cases}$$

Then the associated positive number for permutation $a_1a_2...a_i...a_n$ is given by

$$\prod(a_1a_2...a_n) = 1 + \sum_{j=2}^{n} \pi_j \qquad (2.2)$$

We shall interchangeably use $P_{a_1a_2...a_n}$ or $P_{(\Pi(a_1a_2...a_n))}$ to indicate a processor node associated with permutation address $a_1a_2...a_i...a_n$, in the *n*-star.

C. Routing in the Star graph

We start this section with two definitions and then introduce two useful routing algorithms that we shall use in the last phase of our algorithm.

Definition 1. Let $S_{n-1}(i)$ be the sub graph of S_n induced by all the vertices with the same last symbol *i,* for some $1 \le i \le n$ [2].

A $S_{n-1}(i)$ is an (*n*-1)-star defined on symbols {1,2,…*n*}-{*i* }. Thus, S_n can be decomposed into *n* sub-(*n*-1)-stars, $S_{n-1}(i)$, $1 \le i \le n$. For example, S_4 in Fig 1(c) contains four 3-stars namely $S_3(1)$, $S_3(2)$, $S_3(3)$ and $S_3(4)$.

Definitin 2. Let m_1 and m_2 be two distinct symbols from {1,2,3…*n*}. We use the notation m_1*m_2 to represent a permutation of {1,2,..*n*} whose first and last symbols are m_1 and m_2 respectively with * representing any permutation of the *n*-2 symbols in {1,2,…*n*}-{m_1,m_2}. Similarly, m_1* is a permutation of *n* symbols whose first symbol is m_1 and $*m_2$ is a permutation of *n* symbols whose last symbol is m_2 [2].

Consider the following problem: Given $S_{n-1}(i)$ and $S_{n-1}(j)$, $i \ne j$, it is required to send the contents of the processors in $S_{n-1}(i)$ to the processors $S_{n-1}(j)$. By sending the contents of $S_{n-1}(i)$ to $S_{n-1}(j)$ we mean that the content of each processor in $S_{n-1}(i)$ is routed to a processor in $S_{n-1}(j)$ such that no two processors in $S_{n-1}(i)$ send their to contents the same processor in $S_{n-1}(j)$. This can be accomplished in constant time.as shown in procedure SEND[1].

Procedure SEND (*i*,*j*)
{
for all s=*i **do in parallel**
node P_s sends datum to its neighbour along connection n
end for
for all *w= i*k,* $k \ne j$ **do in parallel**

node p_w sends datum to its neighbour p_{j*k}

end for

for all *v*= *j***k*, $k \neq i$ **do in parallel**

node p_v sends datum to its neighbor along connection *n*

end for

}

Let *I* = $i_1, i_2 \ldots i_l$ and *J*= $j_1, j_2, \ldots j_l$ be two sequences from {1,2,..n} such that no two elements of *I* are equal, no two elements of *J* are equal and no element of *I* is equal to an element of J. It is desired to send the data in $S_{n-1}(i_1)$, $S_{n-1}(i_2)$, … , $S_{n-1}(i_l)$ to $S_{n-1}(j_1), S_{n-1}(j_2) \ldots S_{n-1}(j_l)$ such that the contents of $S_{n-1}(i_k)$ are sent to $S_{n-1}(i_k)$, $1 \leq k \leq l$. This routing can also be achieved in constant time by procedure GROUP_SEND as follows[1].

Procedure GROUP_SEND (I, J)

{

for *k*=1 to *l* **do in parallel**

SEND (i_k, j_k);

end for

}

To develop the parallel algorithm for Hermite interpolation we need the following algorithm (see[4]) to build the Hamiltonian circuits and paths in star graph.

The input to the algorithm is the degree of the star graph and the output is two arrays of integers *Next[1,...n!]* and *Previous[1…n!]* to be discussed in main phase.

Algorithm Hamiltonian_cycle_embedding_in_S*n;*

Input: Degree of the desired star network, *n:* integer;

Output: Arrays *Next*[1..*n*!] and *Previous*[1..*n*!] of integer;

{ C= '123…*n*'*;*

Hamiltonian_circuit ('*n*','123…(*n*-2)(*n*-1)');

};

Procedure Hamiltonian_circuit (*i, list*)

{

If *list*=Ø **then** *Next*[$\tilde{O}(C)$]=*i*; *C*= $G_i(C)$; *Previous*[$\tilde{O}(C)$]=*i*;

Else for *j*=length(*list*) **downto** 1 **do**

Hamiltonian_circuit(*list*[*j*], *list* - *list*[*j*]);

Next[$\tilde{O}(C)$]=*i*; *C*= $G_i(C)$; *Previous*[$\tilde{O}(C)$]=*i*;

end for

Hamiltonian_path(*list*[1], *list* - *list*[1]);

Next[$\tilde{O}(C)$]=*i*; *C*= $G_i(C)$; *Previous*[$\tilde{O}(C)$]=*i*

Hamiltonian_circuit(*list*[1], *list* - *list*[1]);

end if

};

Procedure Hamiltonian_path (*i, list*)

{

If *list*=Ø **then** *Next*[$\tilde{O}(C)$]=*i*; *C*= $G_i(C)$; *Previous*[$\tilde{O}(C)$]=*i*

Else Hamiltonian_circuit(last(*list*), *list* - last(*list*));

Next[$\tilde{O}(C)$]=*i*; *C*= $G_i(C)$; *Previous*[$\tilde{O}(C)$]=*i*

Hamiltonian_circuit(*list*[1], *list* - *list*[1]);

for *j*=1 **to** length(*list*) **do**

Next[$\tilde{O}(C)$]=*i*; *C*= $G_i(C)$; *Previous*[$\tilde{O}(C)$]=*i* Hamiltonian_circuit(*list*[*j*], *list* - *list*[*j*]);

end for

end if

};

Fig.2. The algorithm for embedding a Hamiltonian cycle into S_n

III. THE PARALLEL ALGORITHM

From (1.1) we have

$$H_{2N-1}(x)=\sum_{i=1}^{N}[1-2(x-x_i)l_i^{'}(x_i)][l_i(x)]^2 y_i+\sum_{i=1}^{N}(x-x_i)[l_i(x)]^2 y_i^{'}$$

$$=\sum_{i=1}^{N}h_i(x)l_i^2(x) \qquad (3.1)$$

where $h_i(x)=[1-2(x-x_i)l_i^{'}(x_i)y_i+(x-x_i)y_i^{'}]$

Our proposed parallel algorithm computes *y= f(x)* on a star S_n, given the dataset $(x_1,y_1,y_1^{'})...(x_N,y_N,y_N^{'})$ and the value x, where *N= n!* . The computation in the course of the running of the algorithm is carried out in three phases; initialization, main and final phases. First each processor (node) is given a data point (i.e. ,the *i*th processor is given a data point $(x_i,y_i,y_i^{'})$) which runs parallel with other processors. Each processor $P_i(1\le i\le N)$ calculates $l_i(x)$, $l_i^{'}(x)$ and finally the product $h_i(x)\times l_i^2(x)$ for a given value of x .Then the partial products at all processors is added to obtain the final result, *y=f(x).*

Let us assume each processing node has five registers, R_1,R_2,R_3,R_4 and R_5. In each node , registers R_1 and R_2 will hold terms required to calculate $l_i(x)$ and the register R_5 will hold the term to calculate $l_i^{'}(x_i)$. The registers R_3 and R_4 will be used to implement an all-to-all broadcast algorithm over the Hamiltonian ring embedded in the host network S_n , during the main phase.

We shall use the notation $P_{a_1a_2...a_n}(R_k)$ or $P_{(\prod(a_1a_2...a_n))}(R_k)$, for $1\le k\le 5$, to indicate register R_k in the processor with address $a_1a_2...a_i...a_n$ and $P^{(t)}_{a_1a_2...a_n}(R_k)$ or $P^{(t)}_{(\prod(a_1a_2...a_n))}(R_k)$ to denote register R_k after step *t*. Each step may involve a set of communication and computation operations. The symbol '$\Leftarrow$' denotes a communication operation between two adjacent nodes while '=' indicates a data movement inside the processor.

A. THE INITIALISATION PHASE

In this phase the value of *x, Next[i], Previous[i],* and the data points $(x_i,y_i,y_i^{'})$ are given to the processor P_i , for *i*=1,2,….*N,* to be stored in some locations within the local memory. They may be stored in some registers providing faster access to their contents. Such registers are initialized once in this phase and will not be updated again later. As discussed in section 2., *Next[i]* and *Previous[i]* would be used in the main phase to determine the next and the previous nodes in the embedded Hamiltonian cycle. Then , registers R_1-R_5 of each processor are set to their initial values by the following instruction sequence, for *i=1,2…N.* in parallel:

$P^{(o)}_{(i)}(R_1)=1;$ $\quad P^{(o)}_{(i)}(R_2)=1;$

$P^{(0)}_{i}(R_3)=x_i;$ $\quad P^{(0)}_{i}(R_4)=x_i;$ $\quad P^{(0)}_{i}(R_5)=0;$

B. The Main Phase

The Hermite interpolation algorithm reveals that each node $P_{(i)}$; $1\le i\le N$, needs at some points to broadcast the value of x_i to all other nodes in the network. This all-to-all broadcast is best performed via a ring arrangement of nodes. Since the ring should contain all nodes of the host star network, a Hamiltonian ring embedded in the host network is required . A Hamiltonian ring (also called the Hamiltonian circuit) is a ring embedded in another network (as the host) including all its nodes [12]. To achieve this, we can use any embedding method described in the literature [6,12]. We employ the embedding algorithm introduced in [5] . Fig 2 describes this algorithm that takes the degree of the star network n to produce two arrays, *Next* and *Previous, Next[i]* (*Previous[i])* is the position of the symbol which has the following property: its exchange with the first symbol of the address permutation associated with the *i*th node in the embedded Hamiltonian cycle, will give the address of the next(previous node) of that node in the embedded Hamiltonian cycle. Therefore , if the address of the *i*th node of the Hamiltonian cycle (starting from node 123…*n*) is $a_1a_2...a_n$, the next and the previous nodes are $\Gamma_{Next[i]}(a_1a_2...a_n)$ and $\Gamma_{previous[i]}(a_1a_2...a_n)$. One can easily use the arrays *Next* or *Previous* to construct a Hamiltonian cycle in a *n*-star starting from node 123…*n*. Such a Hamiltonian cycle embedded in a 4-star is illustrated in Fig 1(c) (indicated with hold links).

Since the initialization phase uses these arrays, they should be set to their proper values before it commences. During the initialization phase each node P_i , for $1 \le i \le N$, receives its appropriate data, including the values *Next[i]* and *Previous[i]*.

Then in the main phase, each node uses these address values to communicate with the next and previous nodes in the embedded Hamiltonian cycle. The main phase computes terms $h_i(x)$ for *i=1,2,..N*. To this end, first, all processors with permutations $a_1 a_2 ... a_n$ $(\prod(a_1 a_2 ... a_n) = i)$ perform the following instruction sequence simultaneously.

Parallel algorithm to calculate $h_i(x) \times l_i^2(x)$ for *i=1,2,.....N.*

for t= 0,1….. *N/2*

$$P_{(i)}^{(t+1)}(R_3) \Leftarrow P_{\Gamma(Next([i])}^{(t)}(a_1 a_2 ... a_3)(R_3);$$

$$P_{(i)}^{(t+1)}(R_4) \Leftarrow P_{\Gamma(\Pr evious[i])}^{(t)}(a_1 a_2 ... a_3)(R_4);$$

$$P_{(i)}^{(t+1)}(R_1) = P_{(i)}^{(t)}(R_1) \times (x - P_{(i)}^{(t+1)}(R_3)) \times (x - P_{(i)}^{(t+1)}(R_4));$$

$$P_{(i)}^{(t+1)}(R_2) = P_{(i)}^{(t)}(R_2) \times (x_i - P_{(i)}^{(t+1)}(R_3)) \times (x_i - P_{(i)}^{(t+1)}(R_4));$$

$$P_{(i)}^{(t+1)}(R_5) = P_{(i)}^{(t)}(R_5) + \frac{1}{(x_i - P_{(i)}^{(t+1)}(R_3))} + \frac{1}{(x_i - P_{(i)}^{(t+1)}(R_4))};$$

end for

$$P_{(i)}^{(N/2)}(R_3) \Leftarrow P_{\Gamma(Next[i])}^{(N/2-1)}(a_1 a_2 ... a_3)(R_3);$$

$$P_{(i)}^{(N/2)}(R_1) = P_{(i)}^{(N/2)}(R_1) \times (x - P_{(i)}^{(N/2)}(R_3));$$

$$P_{(i)}^{(N/2)}(R_2) = P_{(i)}^{(N/2-1)}(R_2) \times (x_i - P_{(i)}^{(N/2)}(R_3));$$

$$P_{(i)}^{(N/2)}(R_5) = P_{(i)}^{(N/2-1)}(R_5) + \frac{1}{(x_i - P_{(i)}^{(N/2)}(R_3))};$$

Note that in the last iteration , instructions are changed to avoid multiplying the terms $(x - x_{N/2})$ and $(x_{\Pi(a_1 a_2 ... a_n)} - x_{N/2})$ twice (to R_1 and R_2, respectively). The data path used in these $N/2$ steps is the embedded Hamiltonian ring in which the values $x_1, x_2 ... x_N$ rotate using an all-to-all broadcast method described in [10]. According to this broadcast method, rotating data values stored in R_3 and to the right (increasing processor order) over the ring while data values stored in R_4 and are rotated in the opposite direction (decreasing processor order) provides each processor a copy of the data stored in other processors after $N/2$ rotation steps.

Each steps (from 0 to $N/2$ **-2**) consists of one data communication, (note that the first two communication, instructions can be realized in parallel because of bi-directional links between nodes), four subtractions four multiplications, two divisions and two additions. whereas after the for loop that is at $N/2$ th step, there is one data communication, two subtractions, one addition and one division.

Now to this end all the processors shall execute the following instructions to calculate the *i*-th term($1 \le i \le N$) of the Hermite interpolation.

$$P_{(i)}^{(N/2+1)}(R_1) = \frac{P_{(i)}^{(N/2)}(R_1)}{P_{(i)}^{(N/2)}(R_2)};$$

$$P_{(i)}^{(N/2+1)}(R_5) = P_{(i)}^{(N/2)}(R_5);$$

$$P_{(i)}^{(N/2+2)}(R_1) = P_{(i)}^{(N/2+1)}(R_1) \times P_{(i)}^{(N/2+1)}(R_1);$$

$$P_{(i)}^{(N/2+2)}(R_5) = P_{(i)}^{(N/2+1)}(R_5);$$

$$P_{(i)}^{(N/2+3)}(R_1) = P_{(i)}^{(N/2+2)}(R_1) \times [\{1-2(x-x_i)P_{(i)}^{(N/2+2)}(R_5)\}y_i + (x-x_i)y_i^{'}];$$

Therefore at the end of this phase, we have

$P_{(i)}(R_1) = h_i(x) \times l_i^2(x)$ for $1 \le i \le N$.

Analysis of Computation Statistics

Let M, S, A and D represent the number of multiplications, subtractions, additions and divisions respectively. At the end of ($N/2+3$)th step, there are $N/2$ number of data communications and at the end of $N/2$ th step

$$M = 4(N/2-1)+2 = 2N-2$$
$$S = 2N-4+2 = 2N-2$$
$$A = N-2+1 = N-1$$
$$D = N-2+1 = N-1$$

At the end of $(N/2+2)$ th step

$$M = 2N-2+1 = 2N-1$$
$$S = 2N-2+0 = 2N-2$$
$$A = N-1+0 = N-1$$
$$D = N+0 = N$$

At the end of $(N/2+3)$ th step

$$M = 2N-1+7 = 2N+6$$
$$S = 2N-2+3 = 2N+1$$
$$A = N-1+1 = N$$
$$D = N+0 = N$$

Therefore in mainphase, there are *N/2* data communications , $2N+6$ multiplications, $2N+1$ subtractions , N additions and N divisions. The analysis of the algorithm in main phase shows that there are *N/2* data communications, *N* additions, $2N+1$ subtractions, $2N+6$ multiplications and N divisions.

The final phase:

In this phase, the contents of the registers R_1 in all nodes are added together to obtain the final result. To this end , let us first define two useful bulk accumulation procedures, namely ACCUMULATE and GROUP_ACCUMULATE, adopting the strategy used in the previously defined procedures SEND and GROUP_SEND, as follows.

Procedure ACCUMULATE (i, j, m, n)
{

for all $s = a_1 a_2 ... a_{m-1} i \sigma_{m,n}, (a_l \in \{1,2,...,n\} - \{j\})$ **do in parallel**

$$P_{ia_2 a_3 ... a_{m-1} a_1 \sigma_{m,n}}(R_2) \Leftarrow P_s(R_1);$$

end for

for all $w = ia_1 a_2 ... a_{m-2} k \sigma_{m,n}, (a_l, k \in \{1,2,...,n\} - \{i, j\})$ **do in parallel**

$$p_{ja_2 ... a_{l-1} ia_{l+1} ... a_{m-1} k \sigma_{m,n}}(R_2) \Leftarrow P_w(R_2);$$

end for

for all $v = ja_1a_2...a_{m-2}k\sigma_{m,n},\ (a_l, k \in \{1,2,...,n\} - \{i, j\})$ **do in parallel**

$$P_{ka_1a_2...a_{m-2}j\sigma_{m,n}}(R_2) \Leftarrow P_v(R_2);;$$
$$P_{ka_1a_2...a_{m-2}j\sigma_{m,n}}(R_1) \Leftarrow P_{ka_1a_2...a_{m-2}j\sigma_{m,n}}(R_2) + P_{ka_1a_2...a_{m-2}j\sigma_{m,n}}(R_1);$$

end for

}

Procedure GROUP_ACCUMULATE (**I, J,** *m ,n*)

{ **for** *k*=1 **to** *m/2* **do in parallel ACCUMULATE** $(i_k, j_k, m, n);;$

end for

}

where $\sigma_{m,n} = \begin{cases} (m+1)m...n & m < n, \\ Null & m = n. \end{cases}$

Let procedure Split_IJ divide a set, say **IJ**={ $a_1, a_2, ..., a_l$ }, into two sub-sets **I**={ $a_1, a_2, ..., a_{l/2}$ } and **J**= { $a_{l/2+1} a_{l/2+2} ..., a_{2(l/2)}$ }, where the division operator means an integer division which results in an integer number. Note that **I** ∪ **J** = **IJ** if l is an even number , **I** ∪ **J** = **IJ** -{ a_l } if l is odd, and **I** ∩ **J**={ } . Now in the final phase, using these procedures the contents of register R_1 in all processors are accumulated in register $P_{12...n}(R_1)$. This is realized in (*n*-1) sub phases. After each sub phase s, *s*=1, 2,…,*n*-1, the size of accumulation problem will be reduced by one. Thus the problem will reduce from accumulation in (*n*-*s*+1)-star to accumulation in a (*n*-*s*)-star.

```
for  i = n  down to 2 do
      IJ= {1,2,…,i };
      while( IJ ≠ { i} ) do
            Split_IJ(IJ, I, J);
            GROUP_ACCUMULATE(I, J, i, n)
      end while
end for
```

$$P_{123...n}(R_1) = P_{123...n}(R_2) + P_{123...n}(R_1);$$

In total , this phase includes $\sum_{i=2}^{n} \lceil \log i \rceil$ additions and $1+3\sum_{i=3}^{n} \lceil \log i \rceil$ communication operations.

IV. CONCLUSION

In this paper, an optimal parallel algorithm for computing an N=n! point Hermite interpolation has been introduced. The algorithm consists of three phases. The first phase initializes the processors, the second phase calculates $h_i(x) \times l_i^2(x)$, $1 \le i \le N$ (see eqn (3.1)) optimally using an all-to-all broadcast in $O(N)$ time. The last phase gathers the terms $h_i(x) \times l_i^2(x)$ hold in processors P_i , $1 \le i \le N$, and accumulates them in processor $P_{123...n}$ with a running time of $O(n \log n)$ resulting in total running time $O(N)$.

REFERENCE

[1] Akl S., "Parallel computation: Models and methods, PHI, 1997.

[2] Akl S. , Qiu K. , " A novel routing scheme on the star and parallel computing , 19(1), pp. 95-101,1993.

[3] Akl S., Qiu K., Stojmenovic I., " Fundamental algorithms for the star and pancake interconnection networks with applications to computational geometry", Networks, 23, pp. 215-225, 1993.

[4] Akl S.G., MacKenZie L.M, Sarbazi-Azad H. 14th International conf. on Parallel & Distributed processing symposium(TPDPS 00), May 01-05, 2000, Cancom, Mexico.

[5] Akl S., Duprat J., Ferreira A. G., "Building Hamiltonian circuits and paths on star graphs", Fifth SIAM Conf. Discrete Math., Atlanta, June 11-14, 1990.

[6] Akers S.B., Harel D., Krishnamurthy B., "The star graph: an attractive alternative to the n-cube", proc. Int. Conf. Parallel Processing, pp. 393-400, 1987.

[7] Capello P. , Gallopoulos E., Koe C. K., "Systolic computation of interpolation polynomials", computing, 45(2), pp.95-117, 1990.

[8] Fragopoulou P., Akl S., "Parallel algorithm for computing Fourier transform on the star graph", IEEE TPDS, 5(5), pp. 525-531, 1994.

[9] Goertzel B., "Lagrange Interpolation on a tree of processors with ring connections", JPDC, 22 , pp.321-323, 1994.

[10] McKeown, G.P., " Iterated interpolation using a systolic array", ACM Trans, Mathematical software, 12, pp. 162-170, 1986.

[11] Menn A., Somani A. K., " an efficient sorting algorithm for the star graph interconnection network", proc. Int. Conf. Parallel processing, pp.III-1-8, 1990.

[12] Nigam M. , Sahni S., Krishnamurthy B., "Embedding Hamiltonians and hypercubes in star interconnection graphs", proc. Int. Conf. Parallel processing, pp. III-340-343, 1990.

[13] Qiu K. , Akl S., "Load balancing , selection and sorting on the star and pancalce interconnection networks", Parallel Algorithm & applications, 2. pp.27-42, 1994.

[14] Sastry S.S. , "Introductory methods for Numerical Anal." 3e PHI.

[15] Sarbazi-Azad H. , Mackenzie L. M., Ould-khaoua M., " A parallel algorithm for lagrange interpolation on k-ary n-cubes', LNCS 1557, pp.85-95,1999.

[16] Wendroff B., Theoretical Numerical Analysis; Academic press Inc. , 1966.

