Parallel Lagrange Interpolation on the Star Graph

H. Sarbazi-Azad
Dept of Computing Sci.
University of Glasgow
Glasgow, U.K.
hsa@dcs.gla.ac.uk

M. Ould-Khaoua
Dept of Computer Science
University of Strathclyde
Glasgow, U.K.
mohamed@cs.strath.ac.uk

L.M. Mackenzie
Dept of Computing Sci.
University of Glasgow
Glasgow, U.K.
lewis@dcs.gla.ac.uk

S.G. Akl
Dept of Computer & Info. Sci.
Queen's University
Kingston, Ontario, Canada.
akl@cs.queensu.ca

Abstract

This paper introduces a parallel algorithm for computing an $N=n!$-point Lagrange interpolation on an n-star ($n>2$). It exploits several communication techniques on stars in a novel way which can be adapted for computing similar functions. The algorithm is optimal and consists of three phases: initialization, main and final. While there is no computation in the initialization phase, the main phase is composed of $n!/2$ steps, each consisting of four multiplications, four subtractions and one communication operation, and an additional step including one division and one multiplication. The final phase is carried out in $(n-1)$ sub-phases each with $O(\log n)$ steps where each step takes three communications and one addition.

1. Introduction

Interpolation techniques are of great importance in numerical analysis since they are widely used in a variety of science and engineering domains where numerical solution is the only way to predict the value of a tabulated function for new input data. Many methods have been proposed of which Lagrange interpolation is one of the best known techniques. *Lagrange interpolation* [14] for a given set of points (x_1,y_1), (x_2,y_2), … , (x_N,y_N) and a value x is defined as

$$f(x) = \sum_{i=1}^{N} y_i L_i(x) \qquad (1)$$

where the term L_i is called a *Lagrange polynomial*, and is given by

$$L_i(x) = \prod_{j=1}^{N} \frac{x - x_j}{x_i - x_j}, \quad j \neq i \qquad (2)$$

When the number of points, N, is very large, a long computation time and a large storage capacity may be required to carry out the above calculation. To overcome this, several authors have proposed parallel implementations for Lagrange interpolation. For instance, Goertzel [9] has introduced a parallel algorithm suitable for a tree topology with N processors augmented with ring connections. The algorithm requires $N/2+O(\log N)$ steps, each composed of two subtractions and four multiplications. Capello *et al* [7] have described a systolic algorithm using $2N-1$ steps on $N/2$ processors where each step, after implementation, requires two subtractions, two multiplications and one division. More recently, a parallel algorithm has been discussed in [13] which uses a k-ary n-cube, consisting of $O(k^n+kn)$ steps, each with 4 multiplications and subtractions, for an $N=k^n$ node interpolation.

The star graph was proposed in [1] 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 [10], load balancing [12], computational geometry algorithms [4], Fourier transform [8], and communication algorithms [3].

This paper proposes a parallel algorithm for computing Lagrange interpolation on the star. The algorithm combines several communication techniques in a novel way to compute an N-point Lagrange 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 points $(x_i, y_i= f(x_i))$ to be taken into account. Examples of such functions exist in *numerical analysis* and *image and signal 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 abilities of the host network. A network is Hamiltonian if a cycle can be embedded in the network which includes all of its nodes [5,11]. The algorithm relies on all-to-all broadcast communication at some stages 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.

2. Preliminaries

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

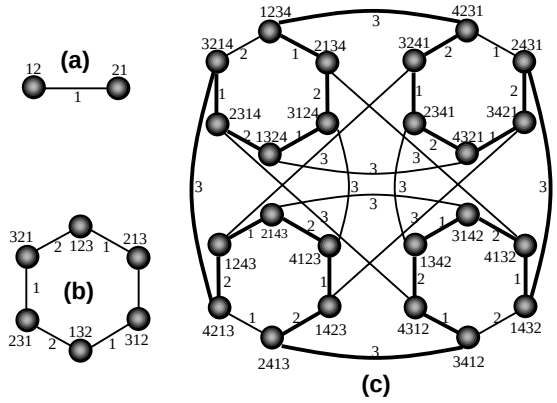

Fig.1- The star graph: (a) 2-star, S_2. (b) 3-star, S_3. (c) 4-start, S_4.

2.1. 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 $a_1a_2\cdots a_{i-1}a_ia_{i+1}\cdots a_n$ is connected to those vertices corresponding to permutations $a_ia_2\cdots a_{i-1}a_1a_{i+1}\cdots a_n$ for $2 \le i \le n$, (that is, those permutations that result from interchanging the first symbol in the permutation $a_1a_2\cdots a_{i-1}a_ia_{i+1}\cdots a_n$ with any of the remaining *n*-1 symbols). The edge connecting the vertex associated with the permutation resulting from the interchange between the 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, $a_1a_2\cdots a_n$, via the *i*th connection as $\Gamma_i(a_1a_2\cdots a_n) = a_ia_2\cdots a_{i-1}a_1 a_{i+1}\cdots a_n$. 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 $N = n!$) [2].

2.2. 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 [2] is used where an ordering, $\prec$, on the nodes of *n*-star is defined as follows. We say $a_1a_2\cdots a_i\cdots a_n \prec b_1b_2\cdots b_i\cdots b_n$ 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, $a_1a_2\cdots a_i\cdots a_n$, we define π_i, for each a_i, with $2 \le i \le n$, to be

$$\pi_i = \left| a_i - i - \sum_{j=i+1}^{n} \langle a_i > a_j \rangle \right| (i-1)! \qquad (3)$$

where $\langle a_i > a_j \rangle = \begin{cases} 1 & \text{if } a_i > a_j \\ 0 & \text{otherwise} \end{cases}$.

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

$$\Pi(a_1a_2\cdots a_n) = 1 + \sum_{j=2}^{n} \pi_j \; . \qquad (4)$$

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

2.3. Routing in the Star graph

This section starts by giving two definitions and then introduces two useful routing algorithms that we shall use in the last phase of our algorithm.

Definition 1. *Let $S_{n-1}(i)$ be the subgraph of S_n induced by all the vertices with the same last symbol i, for some $1 \le i \le n$* [3].

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 $S_3(4)$.

Definition 2. *Let m_1 and m_2 be two distinct symbols from {1,2,…,n}. We use 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* [3].

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 contents the same processor in $S_{n-1}(j)$. This can be accomplished in constant time as shown in Procedure SEND [2].

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

node P_w sends datum to its neighbor 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 $I = \{i_1, i_2, \cdots, i_l\}$ and $J = \{j_1, j_2, \cdots, j_l\}$ 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 $S_{n-1}(i_1), S_{n-1}(i_2), \ldots, S_{n-1}(i_l)$ 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 $S_{n-1}(i_k)$, $1 \le k \le l$. This routing can also be achieved in constant time by Procedure GROUP_SEND as follows [2].

Procedure GROUP_SEND(*I, J*)
{ **for** *k*=1 to *l* **do in parallel**
SEND(i_k , j_k);
end for
};

3. The Parallel Algorithm

The proposed parallel algorithm computes *y=f(x)* on a star, S_n, given the set of points (x_1,y_1), (x_2,y_2), … , (x_N,y_N) and the value *x*, where *N*=*n*!. The computation is carried out in three phases: *initialisation*, *main* and *final* phases. First, the set of points to be interpolated is allocated to the nodes, one point for each node. Then, the Lagrange polynomials, $L_i(x)$, for $1 \le i \le N$, are computed. Finally the sum of the terms $L_i(x) \times y_i$, for $1 \le i \le N$, is calculated to obtain the final result, *y= f(x)*.

Before describing these phases in more details, this section introduces some of the notation which will be used in the development of the algorithm. Let each processing node have four registers, R_1, R_2, R_3 and R_4. In each node, registers R_1 and R_2 will hold the terms required for computing a Lagrange polynomial and 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 notation $P_{a_1a_2\cdots a_n}(R_k)$ or $P_{(\Pi(a_1a_2\cdots a_n))}(R_k)$, for $1 \le k \le 4$, to indicate register R_k in the processor with address $a_1a_2\cdots a_n$ and $P^{(t)}_{a_1a_2\cdots a_n}(R_k)$ or $P^{(t)}_{(\Pi(a_1a_2\cdots a_n))}(R_k)$ 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.

3.1. The initialisation phase

In this phase, the values of *x*, *Next*[*i*], *Previous*[*i*] and the point (x_i, y_i) are given to the processor $P_{(i)}$, for *i*=1,…,*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 initialised once in this phase and will not be updated again later. As will be discussed in the next section, *Next*[*i*] and *Previous*[*i*] will be used in the main phase to determine the next and the previous nodes in the embedded Hamiltonian cycle. Then, registers R_1 - R_4 of each processor are set to their initial values by the following instruction sequence, for all *i*=1,…,*N*, in parallel:

$$P^{(0)}_{(i)}(R_1) = 1; \qquad P^{(0)}_{(i)}(R_2) = 1;$$

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

3.2. The main phase

Examining the communication pattern inherent in the interpolation algorithm reveals that each node $P_{(i)}; (1 \le i < N)$ needs, at some point, to broadcast the value x_i to all the other nodes in the network. This all-to-all broadcast operation is best performed via a ring arrangement of nodes. Since this 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 Hamiltonian cycle or circuit) is a ring embedded in another network (as the host) including all its nodes [11]. To achieve this, we can use any embedding method described in the literature [5,11]. We adapt the embedding algorithm introduced in [5]. Figure 2 shows this algorithm that takes the degree of the star network *n* to produce two arrays, *Next* and *Previous*, where *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 $a_1a_2\cdots a_n$, the next and the previous nodes are $\Gamma_{next[i]}(a_1a_2\cdots a_n)$ and $\Gamma_{previous[i]}(a_1a_2\cdots a_n)$. 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 bold 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 $L_i(x)$ and partial products $L_i(x)\times y_i$ for *i*=1,2,…, *N*. To this end, first, all processors with permutations $a_1a_2\cdots a_n$ ($\Pi(a_1a_2\cdots a_n) = i$) perform the following instruction sequence simultaneously.

for *t*=0,1,…, $N/2-2$ **do**

$$P_{(i)}^{(t+1)}(R_3) \Leftarrow P_{\Gamma_{Next[i]}(a_1a_2\cdots a_n)}^{(t)}(R_3);$$

$$P_{(i)}^{(t+1)}(R_4) \Leftarrow P_{\Gamma_{previous[i]}(a_1a_2\cdots a_n)}^{(t)}(R_4);$$

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

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

end for

$$P_{(i)}^{(N/2)}(R_3) \Leftarrow P_{\Gamma_{Next[i]}(a_1a_2\cdots a_n)}^{(N/2-1)}(R_3);$$

$$P_{(i)}^{(N/2)}(R_1) = P_{(i)}^{(N/2-1)}(R_1)\times\left(x - P_{(i)}^{(N/2)}(R_3)\right);$$

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

Note that in the last iteration, instructions are changed to avoid multiplying the terms $(x - x_{N/2})$ and $(x_{\Pi(a_1a_2\cdots a_i\cdots a_n)} - x_{N/2})$ 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 $x_1,\ldots,x_N$ rotate using an all-to-all broadcasting method described in [6]. According to this broadcast method, rotating data values stored in R_3s to the right (increasing processor order) over the ring while data values stored in R_4s 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 step 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 and four multiplications. The two subtractions in the third instruction could be removed by adding two extra registers to communicate terms $x - x_i$ through the ring as well as x_i because terms $x - x_i$ are computed over and over again by all processors during this phase. It adds just one subtraction in the initialisation phase. Now, as the last step in this phase, all the processors execute the following instruction

$$P_{a_1a_2\cdots a_n}^{(\frac{N}{2}+1)}(R_1) = \frac{P_{a_1a_2\cdots a_n}^{(\frac{N}{2})}(R_1)}{P_{a_1a_2\cdots a_n}^{(\frac{N}{2})}(R_2)}\times y_{\Pi(a_1a_2\cdots a_n)};$$

Therefore, at the end of this phase, we have $P_{(i)}(R_1) = L_i \times y_i$ for $1 \le i \le N$. In the main phase, each processor performs *N/2* data communications (because bi-directional links provide full-duplex communication between two adjacent nodes), 2*N*-1 multiplications, 2*N*-2 subtractions and one division.

3.3. 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= $a_1a_2\cdots a_{m-1}i\sigma_{m,n}$, ($a_l \in \{1,2,\cdots,n\} - \{j\}$) **do in parallel**

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

end for

for all *w*= $ia_1a_2\cdots a_{m-2}k\sigma_{m,n}$,($a_l, k \in \{1,2,\cdots,n\} - \{i, j\}$) **do in parallel**

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

end for

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

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

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

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

Let procedure Split_*IJ* divide a set, say ***IJ***=$\{a_1, a_2, \cdots, a_l\}$, into two sub-sets ***I***= $\{a_1, a_2, \cdots, a_{l/2}\}$ and ***J***=$\{a_{l/2+1}, a_{l/2+2}, \cdots, a_{2(l/2)}\}$, where the division operator means an integer division which results in an integer number. Note that ***I*** $\cup$ ***J*** = ***IJ*** if *l* is an even

```
Algorithm Hamiltonian_cycle_embedding_in_Sn;
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[Π(C)]=i; C=Γi(C); Previous[Π(C)]=i;
  Else for j=length(list) downto 1 do
          Hamiltonian_circuit(list[j], list - list[j]);
          Next[Π(C)]=i; C=Γi(C); Previous[Π(C)]=i;
       end for
       Hamiltonian_path(list[1], list - list[1]);
       Next[Π(C)]=i; C=Γi(C); Previous[Π(C)]=i;
       Hamiltonian_circuit(list[1], list - list[1]);
  end if
};
---------------------------------------------------------
Procedure Hamiltonian_path ( i, list)
{ If list=Ø then Next[Π(C)]=i; C=Γi(C); Previous[Π(C)]=i;
  Else Hamiltonian_circuit(last(list), list - last(list));
       Next[Π(C)]=i; C=Γi(C); Previous[Π(C)]=i;
       Hamiltonian_circuit(list[1], list - list[1]);
       for j=1 to length(list) do
           Next[Π(C)]=i; C=Γi(C); Previous[Π(C)]=i;
           Hamiltonian_circuit(list[j], list - list[j]);
       end for
  end if
};
```

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

number, $\boldsymbol{I} \cup \boldsymbol{J} = \boldsymbol{IJ}$-$\{a_l\}$ if l is odd, and $\boldsymbol{I} \cap \boldsymbol{J}=\{\}$. Now, in the final phase, using these procedures the contents of registers R_1 in all processors are accumulated in register $P_{123\cdots n}(R_1)$. This is realized in n-1 sub-phases. After each sub-phase s, s=1,…,n-1, the size of accumulation problem will be reduced by one. Thus the problem will reduce from accumulation in a (n-s+1)-star to accumulation in a (n-s)-star.

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

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

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. Fig.3 shows the data paths used during this phase for a 24-point interpolation on a S_4. As can be seen in the figure, this phase consists of three sub-phases. In the first sub-phase the algorithm work on S_4 through six steps. The second sub-phase operates on sub-star $S_3(4)$ during 6 steps. Finally, the last sub-phase, sub-phase 3, deals with the sub-star $S_2(34)$, where there is only one step according to the above algorithm. The last instruction, the addition, accumulates the final result, $y=f(x)$, into $P_{123\cdots n}(R_1)$.

4. Conclusion

The star graph has been proposed as an attractive alternative to the hypercube topology for interconnecting processors in parallel computers. It has desirable properties, e.g. lower node degree and diameter than the hypercube, and is both edge-symmetric and vertex-symmetric. In this paper, an optimal algorithm for computing an N=n!-point Lagrange interpolation on an n-star has been proposed combining several techniques already introduced for stars. This method can be easily adapted for computing any similar function, $f(x)$, whose value for a given input x requires all other known data points to be taken into account. Moreover, the method can be employed for other networks which are Hamiltonian since the main phase of the algorithm relies on an all-to-all broadcasting method which requires a Hamiltonian cycle embedded in the host network. However, the performance of the algorithm may vary for different networks as communication abilities vary from one network to another.

The algorithm consists of three phases. In the first phase, the system is initialized. In the second phase, terms $L_i \times y_i$; $1 \le i \le N$, are optimally calculated using an all-to-all broadcast method in $O(N)$ time. The last phase gathers the terms $L_i \times y_i$ hold in processors $P_{(i)}$; $1 \le i \le N$, and accumulates them in processor $P_{123\cdots n}$ with a running time of $O(n \log n)$ resulting in a total running time of $O(N)$. Since the running time for such an interpolation on a single-processor system is of $O(N^2)$, the parallel algorithm proposed here is time-optimal. In total, the algorithm requires $N/2+1+3\sum_{i=3}^{n}\lceil \log i \rceil$ communication operations, $2N$-1 multiplications, $2N$-$2+\sum_{i=2}^{n}\lceil \log i \rceil$ subtractions or additions (if two extra registers are used in each node) and one division, without taking into account any parallelism in the internal architecture of each processor.

References

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

[2] S. Akl, Parallel Computation: Models and methods, Prentice Hall, 1997.

[3] S. Akl, K. Qiu, "A novel routing scheme on the star and pancake networks and its applications", *Parallel. Computing*, 19(1), pp.95-101, 1993.
[4] S. Akl, K. Qiu, I. Stojmenovic, "Fundamentals algorithms for the star and pancake interconnection networks with applications to computational geometry", *Networks*, 23, pp.215-225, 1993.
[5] S. Akl, J. Duprat, A.G. Ferreira, "Building Hamiltonian circuits and paths on star graphs", *Fifth SIAM Conf. Discrete Math*., Atlanta, June 11-14, 1990.
[6] G.P. McKeown, "Iterated interpolation using a systolic array", *ACM Trans. Mathematical software*, 12, pp. 162-170, 1986.
[7] P. Capello, E. Gallopoulos, C.K. Koc, "Systolic computation of interpolation polynomials", *Computing*, 45(2), pp.95-117, 1990.
[8] P. Fragopoulou, S. Akl, "A parallel algorithm for computing Fourier transform on the star graph", *IEEE TPDS*, 5(5), pp. 525-531, 1994.
[9] B. Goertzel, "Lagrange interpolation on a tree of processors with ring connections", *JPDC*, 22, pp.321-323, 1994.
[10] A. Menn, A.K. Somani, "an efficient sorting algorithm for the star graph interconnection network", *Proc. Int. Conf. Parallel Processing*, pp. III-1-8, 1990..
[11] M. Nigam, S. Sahni, B. Krishnamurthy, "Embedding Hamiltonians and hypercubes in star interconnection graphs", *Proc. Int. Conf. Parallel Processing*, pp. III-340-343, 1990.
[12] K. Qiu, S. Akl, "Load balancing, selection and sorting on the star and pancake interconnection networks", *Parallel. Algorithm & applications*, 2, pp. 27-42, 1994.
[13] H. Sarbazi-Azad, L.M. Mackenzie, M. Ould-Khaoua, "A parallel algorithm for Lagrange interpolation on *k*-ary *n*-cubes", *LNCS* 1557, pp. 85-95, 1999.
[14] B. Wendroff, Theoretical Numerical Analysis, Academic Press Inc., 1966.

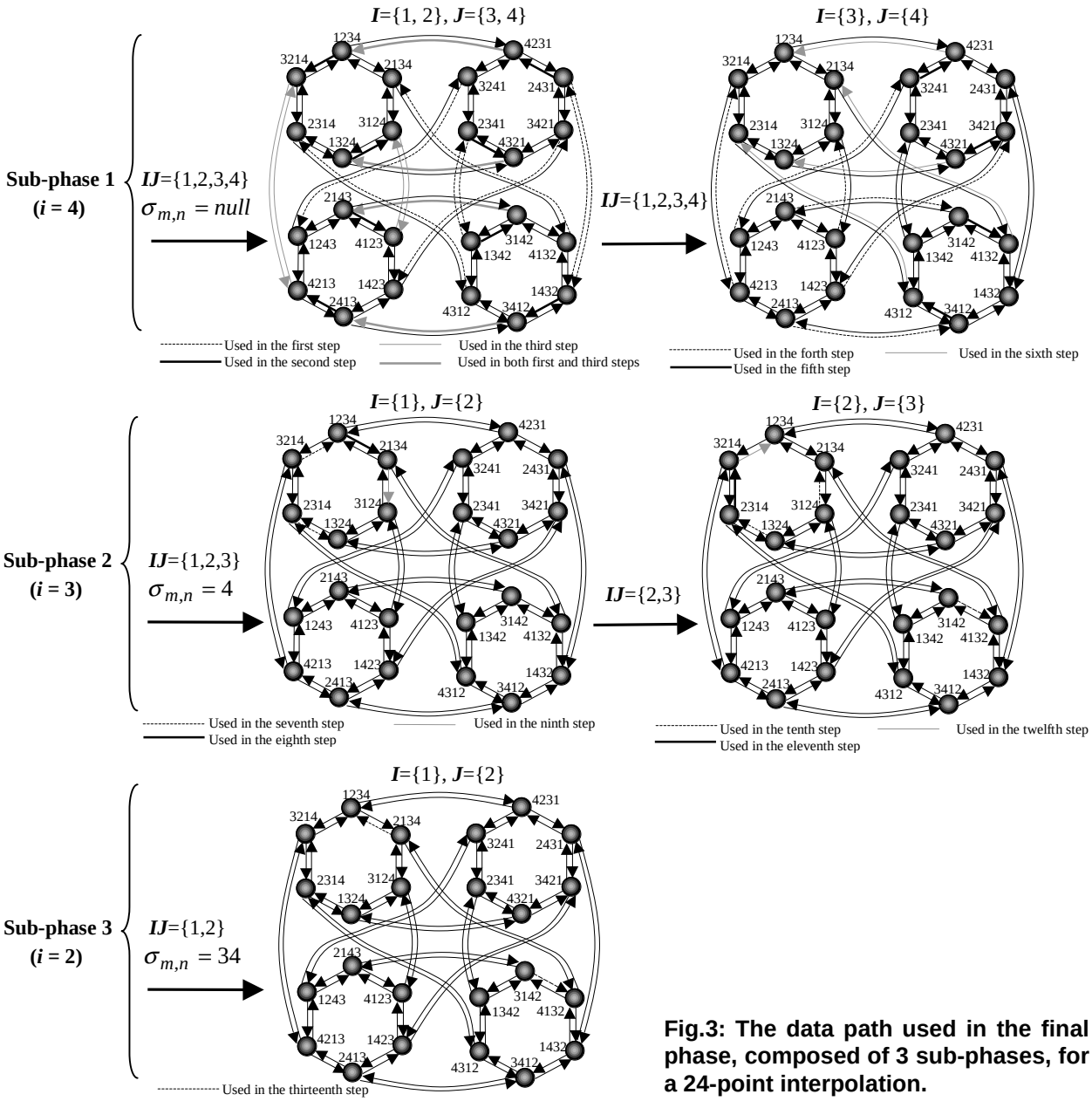

Fig.3: The data path used in the final phase, composed of 3 sub-phases, for a 24-point interpolation.