J Supercomput (2008) 46: 1–14
DOI 10.1007/s11227-008-0179-z

Parallel Lagrange interpolation on k-ary n-cubes with maximum channel utilization

Aminollah Mahabadi · Hamid Sarbazi-Azad · Ebrahim Khodaie · Keivan Navi

Published online: 13 February 2008

Abstract This paper proposes an efficient parallel algorithm for computing Lagrange interpolation on k-ary n-cube networks. This is done using the fact that a k-ary n-cube can be decomposed into n link-disjoint Hamiltonian cycles. Using these n link-disjoint cycles, we interpolate Lagrange polynomial using full bandwidth of the employed network. Communication in the main phase of the algorithm is based on an all-to-all broadcast algorithm on the n link-disjoint Hamiltonian cycles exploiting all network channels, and thus, resulting in high-efficiency in using network resources. A performance evaluation of the proposed algorithm reveals an optimum speedup for a typical range of system parameters used in current state-of-the-art implementations.

Keywords Multicomputers · Interconnection networks · k-ary n-cubes · Link-disjoint Hamiltonian cycles · Parallel algorithms · Lagrange interpolation · Performance analysis

A. Mahabadi
Department of Electrical and Computer Engineering, Shahed University, Tehran, Iran

H. Sarbazi-Azad (✉)
Department of Computer Engineering, Sharif University of Technology, Tehran, Iran
e-mail: azad@sharif.edu

H. Sarbazi-Azad
School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran
e-mail: azad@ipm.ir

E. Khodaie
National Organization for Educational Testing (NOET), Tehran, Iran

K. Navi
Faculty of Electrical and Computer Engineering, Beheshti University, Tehran, Iran

1 Introduction

The n-dimensional torus network (or nD torus for short), including the k-ary n-cube, has been the most popular multicomputer interconnection network due to its desirable properties [5, 8, 9, 11, 12] such as ease of implementation, recursive structure and ability to exploit communication locality found in many parallel applications [2] to reduce message latency [1]. The torus has been used as the underlying topology of many practical systems, such as the Intel/CMU iWarp [18], Cray T3E [14], and Cray T3D [4]. Since many computation problems can be naturally mapped into meshes and tori, the torus network (so called mesh with wraparounds) can be efficiently employed [11]. A cycle including all nodes of a given network and a subset of its links is called a Hamiltonian cycle of that network. Hamiltonian cycles are widely used in many graph algorithms [7] including collective communication routing strategies [16].

Existence of multiple link-disjoint Hamiltonian cycles in networks is very important in fault-tolerant parallel computing as it permits, for example, designing communication algorithms (including collective communication) in the presence of link failure. The problem of finding link-disjoint Hamiltonian cycles has widely been addressed for k-ary n-cubes in the literature. The fact that a 2D torus contains two link-disjoint Hamiltonian cycles has widely been addressed in the literature. For instance, in [6], it has been shown that any $k^m \times k$ torus, $k > 2$, $m = 1, 2, \ldots$, contains two link-disjoint Hamiltonian cycles while in [15] a general $k_1 \times k_2$ torus, $k_1, k_2 > 2$, has been considered and decomposed into two link-disjoint Hamiltonian cycles. A k-ary n-cube, where $n = 2^l$ $(l = 1, 2, \ldots)$ has been shown to have n link-disjoint Hamiltonian cycles [6]. Latifi and Zheng [15] have shown that any nD torus where the size of each dimension is a multiple of 4, i.e., $k_i = 4m_i$, $1 \leq i \leq n$, can be decomposed into n link-disjoint Hamiltonian cycles. In their conclusion, they have conjectured the existence of n link-disjoint Hamiltonian cycles in any nD torus leaving it as an open problem to be proved in future research. However, through a thorough search in the literature, we noticed that this problem was addressed in [3], where it is proven that an nD torus can be decomposed into n link-disjoint Hamiltonian cycles.

Polynomial interpolation techniques are of great importance in science and engineering. Many polynomial interpolation techniques have been proposed in the past of which Lagrange interpolation method is of the most known and used ones. For this interpolation technique, we must first calculate Lagrange polynomials build on the input points. When there are a large number of points, a large storage capacity and long computation time may be required to carry out the interpolation polynomials. To overcome this, several authors have recently proposed parallel implementations for Lagrange interpolation. For instance, Goertzel [13] 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, Gallopoulos, and Koc [10] have described another algorithm using $2N - 1$ steps on $N/2$ processors, where each step requires two subtractions, two multiplications and one division.

Since k-ary n-cube networks have been the underlying topology of a number of practical parallel machines, more recently in [20], a parallel algorithm has been proposed to realize Lagrange interpolation on a k-ary n-cube. The algorithm is optimal but due to inefficient use of channels, its performance is very sensitive to network communication latency.

In this paper, we first review the problem of finding link-disjoint Hamiltonian cycles in the nD torus. We then use the n link-disjoint Hamiltonian cycles of a k-ary n-cube multicomputer to efficiently compute an nk^n-point Lagrange interpolation. This algorithm relies on broadcast communication at some stages during computation, as will be shown later. This is achieved by concurrently using the n link-disjoint Hamiltonian cycles embedded in the host k-ary n-cube to ensure the best possible utilization of network channels.

The rest of the paper is organized as follows. Section 2 introduces the preliminaries and background required to understand the next sections. In Sect. 3, we review the problem of decomposing an nD torus (and thus, a k-ary n-cube as a special example of nD torus) into n link-disjoint Hamiltonian cycles. Section 4 introduces the parallel algorithm based on the embedded link-disjoint Hamiltonian cycles. In Sect. 5, the performance of the proposed algorithm is evaluated. Finally, Sect. 6 draws some conclusions from this study.

2 Preliminary and background

This section gives some definitions and notations that help better understanding the next section. The related work in this line of research is also briefly surveyed.

2.1 The nD torus and k-ary n-cube

An nD $k_1 \times k_2 \times \cdots \times k_n$ torus, $T_{k_1,k_2,\ldots,k_n}$, has $N = \prod_{i=1}^{n} k_i$ nodes arranged in n dimensions with k_i node at dimension i, $1 \leq i \leq n$. Node A in $T_{k_1,k_2,\ldots,k_n}$ is labeled with a distinct n-digit mixed-radix vector $[a_1, a_2, \ldots, a_n]$, where $a_i, 0 \leq a_i \leq k_i - 1, 1 \leq i \leq n$, indicates the position of the node in the ith dimension. In a torus $T_{k_1,k_2,\ldots,k_n}$, defined over radix vector $K = [k_1, k_2, \ldots, k_n]$, two nodes $A = [a_1, a_2, \ldots, a_n]$ and $B = [b_1, b_2, \ldots, b_n]$ are interconnected if and only if there is an $i, 1 \leq i \leq n$ such that $a_i = b_i \pm 1 \pmod{k_i}$ and $a_j = b_j$ for $1 \leq j \leq n, j \neq i$. Thus, in $T_{k_1,k_2,\ldots,k_n}$ each node is adjacent with two nodes in each dimension, hence $2n$ nodes in total. A k-ary n-cube, C_k^n, is a variation of torus where each dimension i, $1 \leq i \leq n$, is of size k; i.e., $C_k^n = T_{\underbrace{k,k,\ldots,k}_{n}}$. Figure 1 shows some examples of torus and k-ary n-cube networks.

2.2 Lagrange interpolation

Lagrange interpolation for a given set of points (x_m, y_m) $(0 \leq m \leq N-1)$ and value x is carried out using the formula [21]

$$f(x) = \sum_{m=0}^{N-1} L_m(x) y_m \tag{1}$$

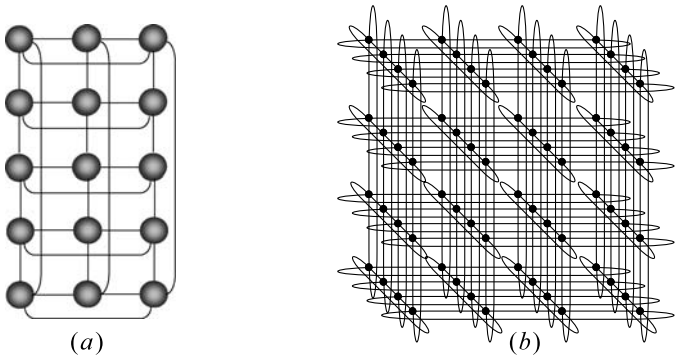

Fig. 1 Examples of tori, (**a**) 5×3 torus $T_{5,3}$, and (**b**) 4-ary 3-cube, C_4^3

where $L_m(x)$ is called Lagrange polynomial, and is given by

$$L_m(x) = \frac{(x - x_0) \cdots (x - x_{m-1})(x - x_{m+1}) \cdots (x - x_{N-1})}{(x_m - x_0) \cdots (x_m - x_{m-1})(x_m - x_{m+1}) \cdots (x_m - x_{N-1})}. \quad (2)$$

3 Hamiltonian decomposition of the *n*D torus

In this section, we review the results reported in the literature on the Hamiltonian decomposition of the nD torus (and consequently, any k-ary n-cube). We then use it to propose a communication-efficient parallel algorithm for computing Lagrange interpolation polynomials on k-ary n-cubes, in Sect. 4.

Let $G = (V, E)$ be an undirected graph of N nodes defined over vertex set V and edge set E. G is called Hamiltonian if it contains a Hamiltonian cycle.

Hamiltonian cycle: Let $H = \{A_1, A_2, \ldots, A_r\}$ be a sequence of node addresses in network G. H is called a cycle (or ring) of length r if (a) $A_i \neq A_j$ for $1 \leq i, j \leq r$, (b) $D(A_i, A_{i+1}) = 1$ for $1 \leq i \leq (r-1)$, and (c) $D(A_1, A_r) = 1$, where $D(A_i, A_j)$ gives the distance (i.e., the number of hops) between two nodes with addresses A_i and A_j. If $r = N$, where N is the number of nodes in G, then H is called a Hamiltonian cycle (or ring) of G. It is clear that $H = (V_H, E_H)$ is a sub graph of G and we have $V_H = V$, $E_H \subseteq E$, and $|E_H| = N$.

Link-disjoint Hamiltonian cycles: Suppose $H_1 = (V_1, E_1), H_2 = (V_2, E_2), \ldots,$ and $H_m = (V_m, E_m)$ are m different Hamiltonian cycles of G. $H_i = (V_i, E_i), 1 \leq i \leq m$, are link-disjoint if $E_i \cap E_j = \phi$ for all $1 \leq i, j \leq m, i \neq j$, and $\bigcup_{i=1}^{m} E_i \subseteq E$.

It is easy to see that the upper bound for *m,* the number of link-disjoint Hamiltonian cycles in a network $G = (V, E)$, is $\frac{|E|}{|V|}$. When G is decomposed into the maximum possible number of link-disjoint Hamiltonian cycles $M = |E|/|V|$, we have a perfect Hamiltonian decomposition where $E_i \cap E_j = \phi$ for all $1 \leq i, j \leq M, i \neq j$, and $\bigcup_{i=1}^{M} E_i = E$. Hereafter in this paper, by Hamiltonian decomposition we mean a perfect Hamiltonian decomposition.

Note that the maximum possible number of link-disjoint Hamiltonian cycles, M, for the nD torus (hence, the k-ary n-cube) is n.

Theorem 1 [3] *Any nD torus, $T_{k_1,k_2,\ldots,k_n}$, $k_i > 2$, $1 \leq i \leq n$, can be decomposed into n link-disjoint Hamiltonian cycles.*

A straightforward and clear proof for Theorem 1 based on product networks was also given in [19].

Since $C_k^n = T_{\underbrace{k,k,\ldots,k}_{n}}$, the following corollary can be directly resulted from Theorem 1.

Corollary 1 [19] *A k-ary n-cube, C_k^n, can be decomposed into n link-disjoint Hamiltonian cycles.*

4 The proposed parallel algorithm

The proposed algorithm consists of three phases. In the initialization phase, the registers in each processor are set to their initial values. In the main phase, the Lagrange polynomials, $L_m(x), 0 \leq m \leq nN-1$, are computed on an $N(=k^n)$-node k-ary n-cube network. In the final phase, the sum of the terms $L_m(x) \times y_m$ is calculated to produce the final result $y = f(x)$ using (1). Before describing each phase in more detail, let us introduce some useful notation.

Let the n link-disjoint Hamiltonian cycles embedded in the host k-ary n-cube be denoted as $H_0, H_1, \ldots, H_{n-1}$. Let each processor have a register R_1 and n set of 3 registers, where set $i, 0 \leq i < n$, used for communication and calculation within Hamiltonian cycle H_i. Let the 3 registers used in set i be $R_{i,2}$, $R_{i,3}$ and $R_{i,4}$. Registers R_1 and $R_{i,2}$ are used to compute the terms required for evaluating Lagrange polynomial $L_m(x)$ while registers $R_{i,3}$ and $R_{i,4}$ are used as buffers during an all-to-all broadcast communication in the network. Let $P_{i_1,i_2,\ldots,i_n}(R_{i,m})$ denote the content of register $R_{i,m}, 2 \leq m \leq 4$, in processor $P_{i_1,i_2,\ldots,i_n}$. $P_{i_1,i_2,\ldots,i_n}$ is the processor located at position i_1 in dimension 1, i_2 in dimension 2, $\ldots$, and i_n in dimension n. Also, let $A_{i_1,i_2,\ldots,i_n}$ indicate the linear address of the processor $P_{i_1,i_2,\ldots,i_n}$, i.e.,

$$A_{i_1,i_2,\ldots,i_n} = \sum_{j=1}^{n} i_j k^{j-1}. \tag{3}$$

Also let symbol '$\leftarrow$' indicate data movement inside a processor and symbol '$\Leftarrow$' indicates communication operation between two adjacent processors.

4.1 The initialization phase

Each processor $P_{i_1,i_2,\ldots,i_n}, 0 \leq i_m \leq k-1, 1 \leq m \leq n$, holds the values x and n points $(x_{nA_{i_1,i_2,\ldots,i_n}+i}, y_{nA_{i_1,i_2,\ldots,i_n}+i})$, $0 \leq i < n$. The distribution of points $(x_{nA_{i_1,i_2,\ldots,i_n}+i}, y_{nA_{i_1,i_2,\ldots,i_n}+i})$, $0 \leq i < n$ and x to the processors is not discussed here for brevity.

This can be realized in a pipelined fashion in N communication steps using the n embedded Hamiltonian cycles of N processors. In this phase, registers $R_{i,m}$ of each processor are initialized using the following sequence of instructions:

$P_{i_1,i_2,\ldots,i_n}(R_1) \leftarrow 1;$

for $i = 0, 1, \ldots, n\text{-}1$ **do in parallel**
    $P_{i_1,i_2,\ldots,i_n}(R_{i,2}) \leftarrow 1;$
    $P_{i_1,i_2,\ldots,i_n}(R_{i,3}), P_{i_1,i_2,\ldots,i_n}(R_{i,4}) \leftarrow x_{nA_{i_1,i_2,\ldots,i_n}+i};$
end for;

4.2 The main phase

Examining the communication pattern inherent in the interpolation algorithm reveals that a given processor at some point during computation (as we will see later) needs to broadcast the x-co-ordinate of the points that it holds (i.e.,$x_{nA_{i_1,i_2,\ldots,i_n}+i}$, $0 \le i < n$) to all the other processors. To ensure efficient communication, this all-to-all broadcast operation is best performed according to the ring topology [7]. To achieve this, the algorithm uses in the main phase, n Hamiltonian rings that are embedded in the host k-ary n-cube. Let the functions Next_i and Previous_i when applied to processor $P_{i_1,i_2,\ldots,i_n}$ denote its next and previous processors, respectively, in the embedded ring C_i, $0 \le i < n$.

The terms $L_m(x)$ and partial products $L_m(x) \times y_m$, $0 \le m \le nN-1$, are computed during this phase. To do this, all the processors first perform the following instruction sequence simultaneously.

for $t = 0, 1, \ldots, \lceil N/2 \rceil\text{-}2$ **do**
    for $i = 0, 1, \ldots, n\text{-}1$ **do in parallel**
        $P_{i_1,i_2,\ldots,i_n}(R_{i,3}) \Leftarrow \text{Next}_i\ P_{i_1,i_2,\ldots,i_n}(R_{i,3});$
        $P_{i_1,i_2,\ldots,i_n}(R_{i,4}) \Leftarrow \text{Previous}_i\ P_{i_1,i_2,\ldots,i_n}(R_{i,4});$
    end for;

    for $i = 0, 1, \ldots, n\text{-}1$ **do**
        $P_{i_1,i_2,\ldots,i_n}(R_1) \leftarrow P_{i_1,i_2,\ldots,i_n}(R_1) \times \left(x - P_{i_1,i_2,\ldots,i_n}(R_{i,3})\right) \times \left(x - P_{i_1,i_2,\ldots,i_n}(R_{i,4})\right);$
        for $j = 0, 1, \ldots, n\text{-}1$ **do**
            $P_{i_1,i_2,\ldots,i_n}(R_{i,2}) \leftarrow P_{i_1,i_2,\ldots,i_n}(R_{i,2}) \times \left(x_{nA_{i_1,i_2,\ldots,i_n}+i} - P_{i_1,i_2,\ldots,i_n}(R_{j,3})\right) \times \left(x_{nA_{i_1,i_2,\ldots,i_n}+i} - P_{i_1,i_2,\ldots,i_n}(R_{j,4})\right);$
        end for;
    end for;
end for;

if N is even **then**
  for $i = 0, 1, \ldots, n\text{-}1$ **do in parallel**
    $P_{i_1,i_2,\ldots,i_n}(R_{i,3}) \Leftarrow \text{Next}_i\ P_{i_1,i_2,\ldots,i_n}(R_{i,3})$;
  endfor;

  for $i = 0, 1, \ldots, n\text{-}1$ **do**
    $P_{i_1,i_2,\ldots,i_n}(R_1) \leftarrow P_{i_1,i_2,\ldots,i_n}(R_1) \times \left(x - P_{i_1,i_2,\ldots,i_n}(R_{i,3})\right)$;
    for $j = 0, 1, \ldots, n\text{-}1$ **do**
      $P_{i_1,i_2,\ldots,i_n}(R_{i,2}) \leftarrow P_{i_1,i_2,\ldots,i_n}(R_{i,2}) \times \left(x_{nA_{i_1,i_2,\ldots,i_n}+i} - P_{i_1,i_2,\ldots,i_n}(R_{j,3})\right)$;
    end for;
  end for;
end if;

for $i = 0, 1, \ldots, n\text{-}1$ **do**
  for $j = 0, 1, \ldots, n\text{-}1,\ j \neq i$ **do**
    $P_{i_1,i_2,\ldots,i_n}(R_{i,2}) \leftarrow P_{i_1,i_2,\ldots,i_n}(R_{i,2}) \times (x_{nA_{i_1,i_2,\ldots,i_n}+i} - x_{nA_{i_1,i_2,\ldots,i_n}+j})$;
  end for;
end for;
for $i = 0, 1, \ldots, n\text{-}1$ **do**
  $P_{i_1,i_2,\ldots,i_n}(R_{i,2}) \leftarrow P_{i_1,i_2,\ldots,i_n}(R_1) \times y_{nA_{i_1,i_2,\ldots,i_n}+1} / P_{i_1,i_2,\ldots,i_n}(R_{i,2})$;
  for $j = 0, 1, \ldots, n\text{-}1,\ j \neq i$ **do**
    $P_{i_1,i_2,\ldots,i_n}(R_{i,2}) \leftarrow P_{i_1,i_2,\ldots,i_n}(R_{i,2}) \times (x - x_{nA_{i_1,i_2,\ldots,i_n}+j})$;
  end for;
end for;

$P_{i_1,i_2,\ldots,i_n}(R_1) \leftarrow 0$;
for $i = 0, 1, \ldots, n\text{-}1$ **do**
  $P_{i_1,i_2,\ldots,i_n}(R_1) \leftarrow P_{i_1,i_2,\ldots,i_n}(R_1) + P_{i_1,i_2,\ldots,i_n}(R_{i,2})$;
end for;

At this point during this phase, registers R_1 and R_2 in each processor hold the partial results

$$P_{i_1,i_2,\ldots,i_n}(R_1) = \sum_{i=0}^{n-1} y_{nA_{i_1,i_2,\ldots,i_n}+i} L_{nA_{i_1,i_2,\ldots,i_n}+i}.$$

4.3 The final phase

In this phase, the contents of registers R_1 in all the processors are added together to obtain the final result. This is carried out in n sub-phases, each requiring $\lceil k/2 \rceil$ additions. Starting at dimension n, the content of R_1 in the k processors in that dimension are added together in the following fashion. To increase concurrency, the

processors in that dimension are divided into two groups where the partial summations can be carried out in parallel. The first group contains the processors 0 to $k/2$ and the second one contains processors $k/2+1$ to $k-1$. After $\lceil k/2 \rceil$ steps, each consisting of one data communication and one addition, the result of summation of the all processors in both groups is held in R_1 of the processor whose address is 0 along the dimension n. The same "summation" process is repeated for the subsequent dimension $i = n-1, n-2, \ldots, 1$. In the subphase $n-i+1$, the result is held in R_1 of the processor whose address is 0 along dimension i. After the nth subphase, the final result is accumulated in register R_1 of processor $P_{0,0,\ldots,0}$. The following parallel code shows subphase s $(1 \le s \le n-1)$.

In parallel do
{
  for $i'_{n-s+1} = \lfloor k/2 \rfloor$ **downto** 1 **do**
    for $0 \le i'_{n-s+1}, i'_{n-s}, \ldots, i'_1 < k$ **do in parallel**
      $P_{i'_1, i'_2, \ldots, i'_{n-s+1}-1, 0, \ldots, 0}(R_1) \Leftarrow P_{i'_1, i'_2, \ldots, i'_{n-s+1}-1, 0, \ldots, 0}(R_1) + P_{i'_1, i'_2, \ldots, i'_{n-s+1}, 0, \ldots, 0}(R_1)$;
    end for;
  end for;

  for $i_{n-s+1} = \lfloor k/2 \rfloor + 1$ **to** k-2 **do**
    for $0 \le i_{n-s+1}, i_{n-s}, \ldots, i_1 < k$ **do in parallel**
      $P_{i_1, i_2, \ldots, i_{n-s+1}+1, 0, \ldots, 0}(R_1) \Leftarrow P_{i_1, i_2, \ldots, i_{n-s+1}+1, 0, \ldots, 0}(R_1) + P_{i_1, i_2, \ldots, i_{n-s+1}, 0, \ldots, 0}(R_1)$;
    end for;
  end for;
};

for $0 \le i_{n-s}, i_{n-s-1}, \ldots, i_1 < k$ **do in parallel**
  $P_{i_1, i_2, \ldots, i_{n-s}, 0, \ldots, 0}(R_1) \Leftarrow P_{i_1, i_2, \ldots, i_{n-s}, k-1, 0, \ldots, 0}(R_1) + P_{i_1, i_2, \ldots, i_{n-s+1}, 0, \ldots, 0}(R_1)$;
end for;

After subphase n in this phase, we have $P_{0,0,\ldots,0}(R_1) = \sum_{m=0}^{nN-1} L_m(x) y_m$.

5 Performance analysis

Speedup has often been used to evaluate the performance gains achieved through a parallelisation of a given problem [20]. Speedup is defined as the ratio of the execution time of a program on a single processor to the elapsed time when employing N processors. Hence,

$$S_N = \frac{T_1}{T_N}. \tag{4}$$

For the purpose of our present discussion, let the subtraction and addition operations have the same latency and λ_M and λ_D be the normalized latencies of the multiplication and division operations with respect to that of the addition operation. Consider-

ing (1) and (2), the execution time of an nN-point Lagrange interpolation on a single processor can be written as

$$T_1 = nN\lambda_D + (2n^2N^2 - 3nN)\lambda_M + (n^2N^2 + nN - 1) \tag{5}$$

where the term $2nN(nN-1)\lambda_M + n^2N^2 + nN\lambda_D$ in the above equation accounts for computing Lagrange polynomials (given by (2)) during the main phase and the term $nN\lambda_M + nN - 1$ accounts for computing $y = f(x)$ (according to (1)) during the final phase.

When a k-ary n-cube is used for performing parallel interpolation, as discussed in Sect. 4, the communication overhead to exchange data between processors should also be taken into account. The communication latency in the proposed algorithm is mainly due to the transmission latency of data between neighboring nodes as there is no extra latency due to message blocking in the network given that all interprocessor communication in the algorithm are contension-free. If λ_C is the transmission latency normalized to that of an addition operation, the total execution time for an N-point interpolation is given by

$$\begin{aligned} T_N = {} & \left(2\left\lfloor \frac{N}{2} \right\rfloor (n+n^2) + 2n^2 - n\right)\lambda_M + \left(\left\lfloor \frac{N}{2} \right\rfloor + n\left\lceil \frac{k}{2} \right\rceil\right)\lambda_C + n\lambda_D \\ & + 2\left\lfloor \frac{N}{2} \right\rfloor (n+n^2) + 2n^2 - n + n\left\lceil \frac{k}{2} \right\rceil \end{aligned} \tag{6}$$

where the term $(2\lfloor \frac{N}{2} \rfloor (n+n^2) + 2n^2 - n)\lambda_M + \lfloor \frac{N}{2} \rfloor \lambda_C + n\lambda_D + 2\lfloor \frac{N}{2} \rfloor (n+n^2) + 2n^2 - n$ accounts for the main phase and the term $n\lceil \frac{k}{2} \rceil + n\lceil \frac{k}{2} \rceil \lambda_C$ accounts for the final phase.

A floating-point addition takes 2–3 cycles in the current implementation technology [17]. A floating-point multiplication is about 1.5 times slower than the addition operation, ranging from 3 to 6 cycles [17]. The latency of a floating-point division depends on the actual algorithm used. When the SRT (a subtract-based) algorithm [17] is used, the latency is about 35 cycles (for 64-bit operands). For multiplicative methods (Newton Raphson or binomal series) [17], on the other hand, the latency varies from 12 to 15 cycles. For the purpose of our discussion, we can assume that $1 \le \lambda_M \le 3$ and $4 \le \lambda_D \le 18$. Given the transmission, latency differs from one implementation technology to another and depends also to the channel bandwidth (or width), a wide range of values have been adopted for the parameter λ_C, $1 \le \lambda_C \le 30$.

We have investigated the effects of the parameters λ_D, λ_M, and λ_C on the resulting performance using (1–6).[1] To this end, we have assessed the achieved speedup when the algorithm is run on the k-ary 3-cube (or 3D-dimensional torus) since such low-dimensional k-ary n-cubes are widely used in practical multicomputers including

[1] Please note that the proposed algorithm is synchronous and does not generate unwanted traffic, meaning that the mathematical performance expression derived above are exact and there is no need for validating them.

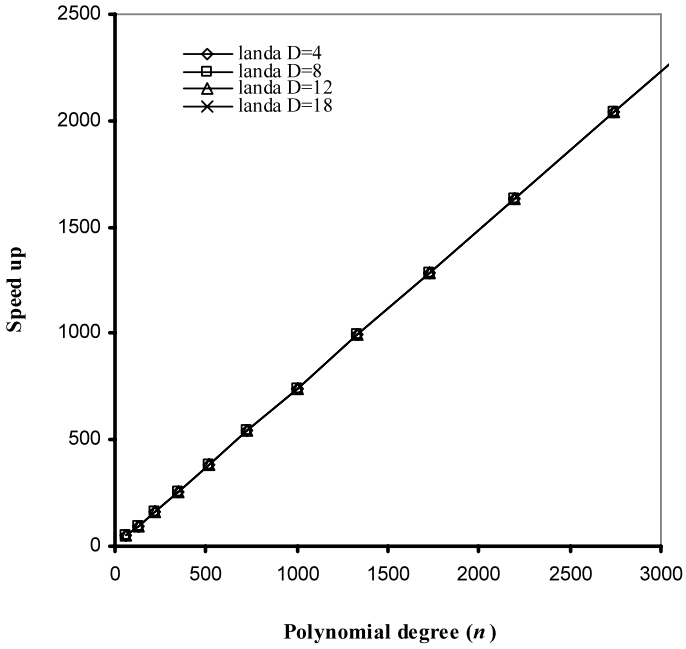

Fig. 2 The effect of Division latency on overall speed up as a function of polynomial degree, n, when $\lambda_C = 1$ and $\lambda_M = 2$

Itel/CMU iWarp, and Cray T3E/T3E [4, 14, 18]. Figure 2 shows performance results against the network size where $\lambda_D = 4, 8, 12$, and 18, and the other parameters are fixed as $\lambda_C = 1$ and $\lambda_M = 2$. The figure reveals that λ_D has hardly any effects on performance.

Figures 3 and 4 assess the impact on performance when the parameters λ_M and λ_C are varied. While Fig. 3 shows that multiplication latency, λ_M, has small effects on performance, Fig. 4 reveals that speed up is sensitive to communication latency, λ_C. For instance, the achieved speedup drops from more than 0.8 when $\lambda_C = 1$ to about 0.6 when $\lambda_C = 30$. It is noteworthy that the algorithm proposed in [20] very much sensitive to the communication latency as a result of its lower network utilization. The performance drop for the algorithm proposed in [20] is from about 0.8 for $\lambda_C = 1$ to about 0.2 when $\lambda_C = 30$.

Figure 5 shows the relative performance of the two algorithms (the one proposed here and the one introduced in [20]) as a function of the number of input points for different communication latency factors. As can be seen in the figure, the new algorithm always exhibits a better performance. This superiority is better noticeable when communication latency λ_C is large. This is due to the higher network channel utilization gained by the new algorithm. Figure 6 shows channel utilization for the new algorithm and the one introduced in [20] for different communication latencies $\lambda_C = 10, 20$ and 30. As can be seen in the figure, the new algorithm utilizes the network channels much better than the algorithm introduced in [20].

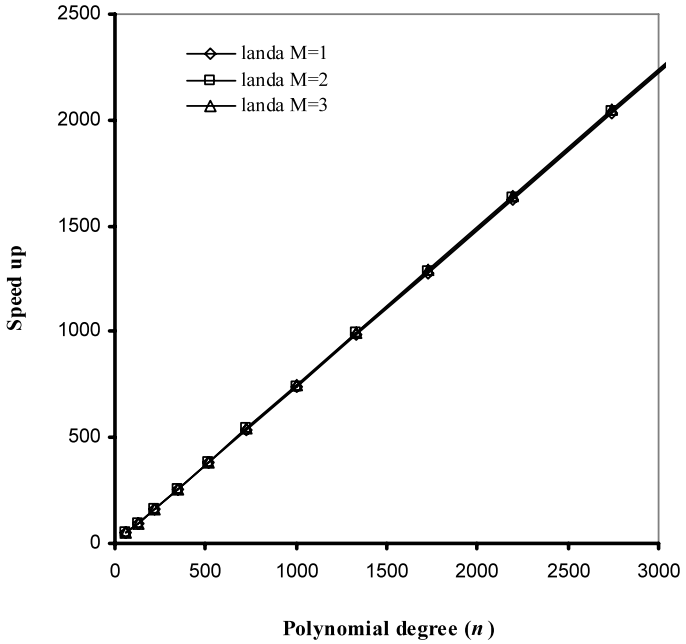

Fig. 3 The effect of Multiplication latency on overall speed up as a function of polynomial degree, n, when $\lambda_C = 1$ and $\lambda_D = 8$

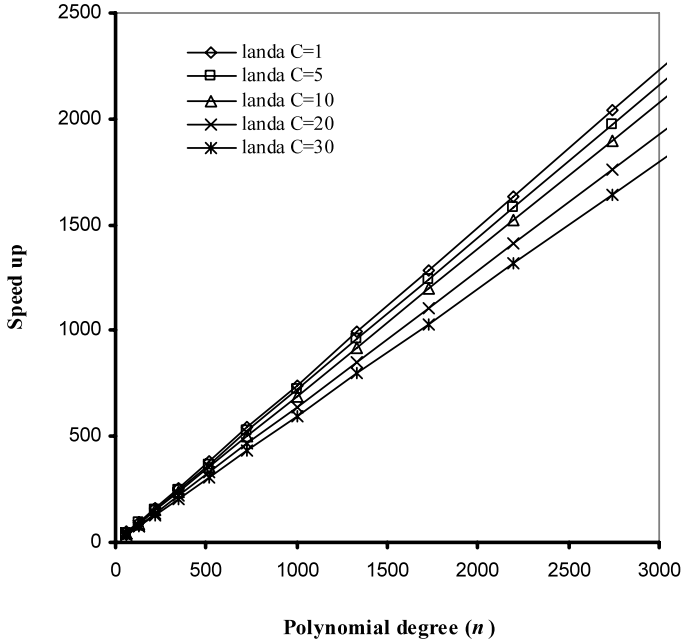

Fig. 4 The effect of Communication latency on overall speed up as a function of polynomial degree, n, when $\lambda_M = 2$ and $\lambda_D = 8$

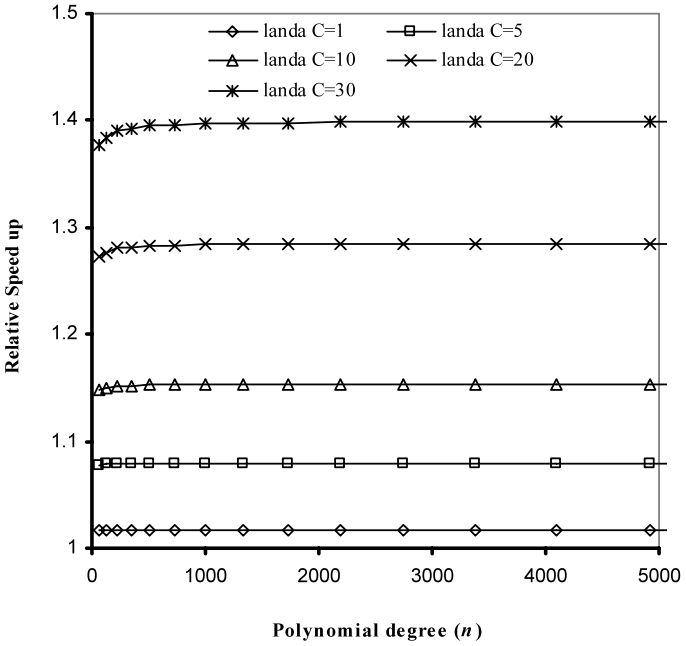

Fig. 5 The effect of Communication latency on relative speed up for the proposed algorithm and the algorithm introduced in [20] as a function of polynomial degree, n, when $\lambda_M = 2$ and $\lambda_D = 8$

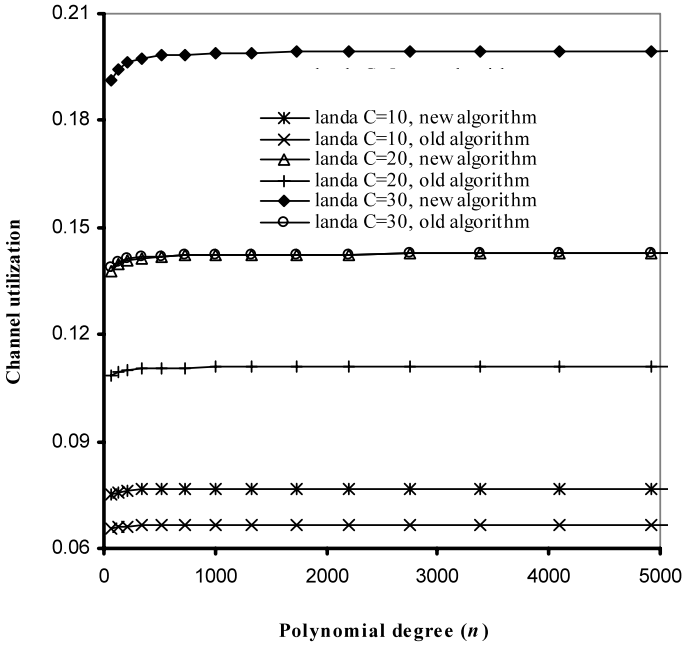

Fig. 6 Channel utilization for the proposed algorithm and the algorithm introduced in [20] as a function of polynomial degree, n, for different communication latency factors, $\lambda_M = 2$ and $\lambda_D = 8$

6 Conclusion and future work

K-ary n-cubes have been widely used in practice due to their desirable properties, such as ease of implementation and ability to exploit communication locality. This paper has proposed a parallel algorithm for Lagrange interpolation on these networks. The algorithm computes an nk^n-point interpolation, requiring $O(n^2k^n)$ multiplications, $O(n^2k^n)$ additions/subtractions and n divisions, without taking into account any parallelisation in the internal computing architecture of the processors, and overlap between computation and communication steps in each node.

Our performance results have revealed that the proposed algorithm has a better performance compared to the latest algorithm introduced in [20]. The new algorithm, proposed here, is less sensitive to the communication latency and can utilize network channels effectively, thus resulting in a superior overall performance. This is done using n link-disjoint Hamiltonian cycles embedded in the host k-ary n-cube network.

Future work in this line can focus on proposing new algorithms for multiple point Lagrange interpolation, and multi-variable (more than one-dimensional) Lagrange interpolation. Developing similar algorithms for other important interpolation techniques using link-disjoint Hamiltonian cycles in k-ary n-cubes and other important network topologies can also be a challenging future research.

References

1. Agrawal A (1991) Limits on interconnection network performance. IEEE Trans Parallel Distributed Syst 2:398–412
2. Al-Ayyoub AE, Day K (1997) Parallel solution of dense linear systems on the k-ary n-cube networks. Int J High Speed Comput 9:85–99
3. Alspach B, Bermond J-C, Sotteau D (1990) In: Hahn G et al (eds) Decomposition into cycles I: Hamilton decompositions. Cycles and rays. Kluwer Academic, Dordrecht, pp 9–18
4. Anderson E, Brooks J, Grassl C, Scott S (1997) Performance of the Cray T3E multiprocessor. In: Proc supercomputing conference
5. Ashir Y, Stewart IA (1997) On embedding cycles in k-ary n-cubes. Parallel Process Lett 7:49–55
6. Bae MM, Bose B (2000) Gray codes for torus and link-disjoint Hamiltonian cycles. In: Proc international parallel and distributed processing symposium (IPDPS'2000), Cancun, 1–5 May, pp 365–370
7. Bertsekas D, Tsitsiklis J (1989) Parallel and distributed computation: numerical methods. Prentice Hall, New York
8. Bettayeb S (1995) On the k-ary hypercube. Theor Comput Sci 140:333–339
9. Bose B, Broeg B, Kwon Y, Ashir Y (1995) Lee distance and topological properties of k-ary n-cubes. IEEE Trans Comput 44:1021–1030
10. Capello P, Gallopoulos E, Koc CK (1990) Systolic computation of interpolation polynomials. Parallel Comput 45:95–117
11. Dally WJ (1990) Performance analysis of k-ary n-cube interconnection networks. IEEE Trans Comput 39:775–785
12. Day K, Al-Ayyoub AE (1997) Fault diameter of k-ary n-cube networks. IEEE Trans Parallel Distributed Syst 8:903–907
13. Goertzel B (1994) Lagrange interpolation on a tree of processors with ring connections. J Parallel Distributed Comput 22:321–323
14. Kessler RE, Schwarzmeier JL (1993) Cray T3D: a new dimension for cray research. In: CompCon, Spring, pp 176–182
15. Latifi S, Zheng SQ (1997) On link-disjoint Hamiltonian cycles of torus networks. Comput Electr Eng 23(1):25–32
16. Mckinley PK, Robinson DF (1995) Collective communication in wormhole-routed massively parallel computers. In: IEEE Computer, Dec 1995, pp 39–50
17. Oberman S (1996) Design issues in high performance floating point arithmetic units. PhD thesis, Stanford Univ

18. Peterson C, Sutton J, Wiley P (1991) iWarp: a 100-MOPS VLIW microprocessor for multicomputers. IEEE Micro 11:26–87
19. Sarbazi-Azad H (2006) Link-disjoint Hamiltonian cycles of k-ary n-cubes. Technical report, Sharif University of Technology, Tehran, Iran
20. Sarbazi-Azad H, Ould-Khaoua M, Mackenzie LM (2001) Employing k-ary n-cubes for parallel Lagrange interpolation. Parallel Algorithms Appl 16:283–299
21. Wendroff B (1966) Theoretical numerical analysis. Academic Press, San Diego

Aminollah Mahabadi received his B.Sc. degree in Computer Engineering (Hardware) from Iran Science and Technology University (Iran), in 1990, and M.Sc. degree in Computer Engineering (Computer Architecture) from Amirkabir University (Iran), in 1996. He is the author of five published books about simulation, and software methodology. His research interests are in ITS, image processing, design and implementation tools for parallel asynchronous systems, and simulation. He currently instructs in the Department of Electrical and Computer Engineering at Shahed University, Iran.

Hamid Sarbazi-Azad received his B.Sc. degree in electrical and computer engineering from Shahid- Beheshti University, Tehran, Iran, in 1992, his M.Sc. degree in computer engineering from Sharif University of Technology, Tehran, Iran, in 1994, and his Ph.D. degree in computing science from the University of Glasgow, Glasgow, UK, in 2002. He is currently associate professor of computer engineering at Sharif University of Technology, and heads the School of Computer Science of the Institute for Studies in Theoretical Physics and Mathematics (IPM), Tehran, Iran. His research interests include highperformance computer architectures, NoCs and SoCs, parallel and distributed systems, performance modeling/evaluation, graph theory and combinatorics, wireless/mobile networks, on which he has published over 190 refereed conference and journal papers. Dr. Sarbazi-Azad has served as the editor-in-chief for the CSI Journal on Computer Science and Engineering since 2005, editorial board member and guest editor for several special issues on high-performance computing architectures and networks (HPCAN) in related journals. Dr. Sarbazi-Azad is a member of ACM and CSI (computer Society of Iran).

Ebrahim Khodaie received his B.Sc. degree from Beheshti University, Tehran, Iran, in 1991, his M.Sc. in Applied Statistics from Tarbiat Modarres University, Tehran, Iran, in 1995, and his Ph.D. in Statistics from Southampton University, Southampton, UK, in 2005. He is now an assistant professor of statistics in the National Organization for Educational Testing (NOET), Tehran, Iran. His research interests include survey sampling, multivariate analysis, order statistics, imputation, data mining, information processing, and neural networks.

Keivan Navi received his B.Sc. and M.Sc. in Computer Hardware Engineering from Beheshti University in 1987 and Sharif University of Technology in 1990, respectively. He also received his Ph.D. in Computer Architecture from Paris XI University in 1995. He is currently associate professor in faculty of electrical and computer engineering of Beheshti University. His research interests include VLSI design, Single Electron Transistors (SET), Carbon Nano Tube, Computer Arithmetic, Interconnection Network, and Quantum Computing. He has published over 30 journal papers and over 70 conference papers.