Fast Computation of Divided Differences and Parallel Hermite Interpolation

Ömer Eğecioğlu[*]
Department of Computer Science, University of California Santa Barbara
Santa Barbara, California 93106

E. Gallopoulos [†]
Center for Supercomputing Research and Development and
Department of Computer Science,
University of Illinois at Urbana-Champaign, Urbana, Illinois 61801

Çetin K. Koç
Department of Electrical Engineering, University of Houston
Houston, Texas 77204

Running head: PARALLEL HERMITE INTERPOLATION
Please address correspondence to Ö. Eğecioğlu

[*]Supported in part by the National Science Foundation under Grant No. NSF DCR-8603722.

[†]Supported by the National Science Foundation under Grants No. US NSF MIP-8410110, US NSF DCR85-09970, and US NSF CCR-8717942, and by AT&T under Grant AT&T AFFL67Sameh.

Abstract

We present parallel algorithms for fast polynomial interpolation. These algorithms can be used for constructing and evaluating polynomials interpolating the function values and its derivatives of arbitrary order (Hermite interpolation). For interpolation, the parallel arithmetic complexity is $O(\log^2 M + \log N)$ for large M and N, where $M - 1$ is the order of the highest derivative information and N is the number of distinct points used. Unlike alternate approaches which use the Lagrange representation, the algorithms described in this paper are based on the fast parallel evaluation of a closed formula for the generalized divided differences. Applications to the solution of dual Vandermonde and confluent Vandermonde systems are described. This work extends previous results in polynomial interpolation and improves the parallel time complexity of existing algorithms.

1 Introduction

Fast algorithms (serial complexity less than $O(N^2)$ and parallel complexity less than $O(N)$ for N input pairs) and asymptotic bounds for polynomial interpolation using as information the value of a function at N distinct points (called simply *interpolation* from here onwards) have been presented by many researchers in the literature [1, 4, 15, 20, 23]. In [8] the authors presented a new algorithm for the fast calculation of the divided difference coefficients of the Newton representation for the interpolating polynomial. The method has parallel complexity[1] $2\lceil \log N \rceil + 2$ and is based on the parallel prefix algorithm[2] ([21] and Appendix A).

In this paper we investigate the more general problem of *Hermite interpolation*, where the input is a set of distinct points and corresponding to each point, prescribed values for a function f and all its derivatives up to some arbitrary order. We show that for large M and N, the computation of the corresponding interpolating polynomial has parallel complexity $O(\log^2 M + \log N)$, where $M-1$ is the order of the highest derivative information and N is the number of distinct points used in the interpolation. Our construction is based on a fast algorithm for the evaluation of all the required polynomial coefficients, the generalized divided differences.

The resulting upper bound extends and improves previous work for polynomial interpolation. Table 1 compares the current computational complexity results for polynomial interpolation.

Non-osculatory				
Representation	Sequential	Ref.	Parallel	Ref.
Lagrange	$O(N \log^2 N)$	[20]	$O(\log N)$	[23, 2]
Newton	$O(N^2)$	[18]	$2 \log N + 2$	[8], Cor. 4.2
Osculatory				
Lagrange-Hermite	$O(n \log n(\log N + 1))$	[4]		
Newton	$O(n^2)$	[29]	$O(s(M) \log N)$	[7]
			$O(\log N + \log^2 M)$	Thm. 4.1

Table 1: Complexity estimates for polynomial interpolation ($s(M)$ is exponential function of M and n is defined in Eq. (1)).

When $N = M$ the interpolation can be done in $O(\log^2 N)$ parallel steps, whereas when $M = 1$ (i.e. no derivatives are involved) the complexity is $O(\log N)$. We show that the algorithm for the latter case of $M = 1$ is identical with the one presented by the authors in [8]. Recently, the authors have presented another parallel algorithm for Hermite interpolation based on algebraic arguments ([7]), which has parallel complexity $O(\log N)$ for M fixed. Nevertheless, as mentioned in that paper, in this case the order of complexity depends exponentially on M, if M is allowed to vary. Consequently, what we present here is a substantial improvement over [7] in terms of theoretical parallel time complexity.

It could be argued that an actual implementation of the proposed algorithm is impractical, since, as is well known, by the time the size of the problem becomes large enough to justify the use of parallelism, polynomial interpolation may break down. We note however that as is

[1]The complexity counts give the number of parallel (elementary) arithmetic operations, which we take to be over the real field for consistency.

[2]All logarithms are base 2.

mentioned in Section 7, certain point arrangements will delay this breakdown. Results in [8] indicate that at least Newton non-osculatory interpolation based on the proposed algorithm for these special points could be of some practical value.

We also remark that the parallel arithmetic complexity of $O(\log^2 M + \log N)$ operations achieved by our algorithm may require a large (but polynomial in the input size) number of processors. Thus its sequential implementation will be less efficient than standard serial algorithms for interpolation. The issues of exact processor count and processor – time tradeoffs for our algorithm are left for future discussion and not addressed here.

Section 2 introduces notation and describes the problem. In Section 3 (Lemma 3.1) the appropriate representation of the GDD (from the point of view of the interpolation algorithm) is introduced. The material in Section 4 culminates in Theorem 4.1, proving the main result. Section 5 contains a brief discussion on polynomial evaluation. Finally, Sections 6 and 7 respectively, contain applications and conclusions.

2 Notation and description of the problem

We are given as input a set of distinct points $\{z_q; q = 0, \ldots, N-1\}$ and for each of these points a set of values $f_q^{(k)}$ with $k = 0, 1, \ldots, p_q - 1$ for $p_q \in \mathcal{Z}^+$, where $\mathcal{Z}^+$ denotes the set of positive integers. We define the *multiplicity vector* $\mathbf{p}$ of the input as

$$\mathbf{p} = (p_0, \ldots, p_{N-1}).$$

Based on this information, we are required to construct a polynomial P of degree $n-1$ where

$$n = \sum_{q=0}^{N-1} p_q \tag{1}$$

such that

$$f_q^{(k)} = P^{(k)}(z_q); \quad k = 0, \ldots, p_q - 1; \; q = 0, \ldots, N-1 \; . \tag{2}$$

Here $P^{(k)}(z_q)$ denotes the derivative of order k of the polynomial P evaluated at the point z_q.

The existence and uniqueness of such a polynomial is well known [5]. For the construction and representation of P two distinct approaches may be followed: the Lagrange–Hermite ([28]) approach and the (generalized) divided difference approach. Here we follow the latter. In the simple case of $p_q = 1$ for all q and $n = N$, the polynomial is written in its Newton form

$$P(s) = \sum_{q=0}^{n-1} f_{[z_0,\ldots,z_q]} \prod_{j=0}^{q-1} (s - z_j) \quad , \tag{3}$$

where the coefficients of the monomial products are the divided differences that are usually constructed recursively by means of tables. These constructions however are sequential in nature and require $O(n)$ parallel arithmetic operations. An alternative method is to use a closed linear formula for each of the divided differences and evaluate them all in parallel by utilizing the properties of the parallel prefix algorithm. With a slight change of wording, a main result of [8] is the following Theorem.

Theorem 2.1 *The divided difference coefficients of the Newton interpolating polynomial for N points can be computed in at most* $2\lceil \log N \rceil + 2$ *parallel arithmetic steps.*

In the general case treated in this paper, there may be more than one datum of information per point z_i. Hence the definition of the divided differences has to be extended to cover this case. This is done by taking the limit of the ratios defining the divided differences for equal arguments. In particular, considering now points $x_0 \leq \cdots \leq x_n$ (not necessarily distinct and coincident in groups with individual z_qs) define

$$f_{[x_q,\ldots,x_{q+k}]} = \frac{f_q^{(k)}}{k!} \tag{4}$$

when $x_q = x_{q+k}$, and

$$f_{[x_q,\ldots,x_{q+k}]} = \frac{f_{[x_{q+1},\ldots,x_{q+k}]} - f_{[x_q,\ldots,x_{q+k-1}]}}{x_{q+k} - x_q} \tag{5}$$

otherwise. These are the *generalized divided differences (GDD)* whose fast evaluation we seek. As with the simpler case of Newton interpolation, the maximum speedup is limited when these definitions are applied directly for the construction of the GDD.

Define a sequence of $n+1$ index-of-multiplicity vectors t_i, each of dimension N as follows: For $0 \leq i \leq p_0$,

$$t_i = (i, 0, \ldots, 0).$$

Otherwise if

$$p_0 < i = p_0 + \ldots + p_{l-1} + \beta \leq p_0 + \ldots p_{N-1} = n$$

for $1 \leq \beta \leq p_l$ and $\beta \in \mathcal{Z}^+$, then

$$t_i = (p_0, \ldots, p_{l-1}, \beta, 0, \ldots, 0) \ .$$

Denote the l^{th} component of t_i by by t_{il}. For each i, let $Q(i)$ denote the smallest index such that $t_{il} = 0$ for $l \geq Q(i)$. Also put $z = (z_0, \ldots, z_{N-1})$ and define

$$z.t_i \quad \equiv \quad (\overbrace{z_0, \ldots, z_0}^{t_{i0}}, \ldots, \overbrace{z_{N-1}, \ldots, z_{N-1}}^{t_{i,N-1}}) \ ,$$

which we will also write as

$$z.t_i \quad = \quad (z_0(t_{i0}), \ldots, z_{N-1}(t_{i,N-1})) \ .$$

Clearly the vector sequence t_i is non decreasing in its components and (componentwise)

$$\begin{aligned} t_i \leq t_n &= (p_0, \ldots, p_{N-1}) \\ &= \mathbf{p} \ . \end{aligned}$$

The vectors t_i provide in increasing order the power index of the factors $(s - z_l)$ in the Newton representation of P. Define

$$w^{(t_i)}(s) = \prod_{l=0}^{N-1} (s - z_l)^{t_{il}} \ , \tag{6}$$

and

$$\begin{aligned} w_q^{(t_i)}(s) &= \frac{w^{(t_i)}(s)}{(s-z_q)^{t_{iq}}} \\ &= \prod_{\substack{l=0 \\ l \neq q}}^{N-1} (s-z_l)^{t_{il}} \quad . \end{aligned}$$

The Hermite interpolating polynomial can then be written in the form

$$P(s) = \sum_{i=1}^{n} f_{[z.t_i]} . w^{(t_{i-1})}(s) \quad . \tag{7}$$

The coefficients $f_{[z.t_i]}$ in Eq. (7) are the GDD and we seek their fast evaluation for $1 \leq i \leq n$.

For example, suppose the interpolation information consists of three distinct points $\{z_0, z_1, z_2\}$ and functional and derivative information corresponding to the multiplicity vector $\mathbf{p} = (2, 1, 3)$. Then $n = 6$ and

$$\begin{pmatrix} t_0 \\ t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \end{pmatrix} = \begin{pmatrix} 000 \\ 100 \\ 200 \\ 210 \\ 211 \\ 212 \\ 213 \end{pmatrix}$$

$$\begin{aligned} w^{(t_0)}(s) &= 1 \\ w^{(t_1)}(s) &= (s-z_0) \\ w^{(t_2)}(s) &= (s-z_0)^2 \\ w^{(t_3)}(s) &= (s-z_0)^2(s-z_1) \\ w^{(t_4)}(s) &= (s-z_0)^2(s-z_1)(s-z_2) \\ w^{(t_5)}(s) &= (s-z_0)^2(s-z_1)(s-z_2)^2 \\ w^{(t_6)}(s) &= (s-z_0)^2(s-z_1)(s-z_2)^3 \quad , \end{aligned}$$

and

$$\begin{aligned} P(s) = {} & f_{[z_0]} + f_{[z_0,z_0]}(s-z_0) + f_{[z_0,z_0,z_1]}(s-z_0)^2 + f_{[z_0,z_0,z_1,z_2]}(s-z_0)^2(s-z_1) \\ & + f_{[z_0,z_0,z_1,z_2,z_2]}(s-z_0)^2(s-z_1)(s-z_2) + f_{[z_0,z_0,z_1,z_2,z_2,z_2]}(s-z_0)^2(s-z_1)(s-z_2)^2 \ . \end{aligned}$$

3 Representation of the Generalized Divided Differences

The elementary definition of the (generalized) divided differences is that they are the coefficients of the Newton representation of the interpolating polynomial. Since a different wording of the objective of polynomial interpolation is to construct a polynomial P which interpolates some function f for which we have functional and derivative information available, we identify the given datum $f_q^{(k)}$ with $\frac{d^k}{dz^k} f(z_q)$. We denote by D^r the differentiation operator applied r times with respect to the underlying variable.

Lemma 3.1 *Let f be analytic in a simply connected region Ω and let C be a rectifiable Jordan curve lying in Ω. Suppose the points z_q for $q = 0, \dots, N-1$ lie in the interior of C. Then the GDDs of f are given by*

$$f_{[z.t_i]} = \sum_{q=0}^{Q(i)-1} \sum_{r=0}^{t_{iq}-1} \frac{f^{(t_{iq}-1-r)}(z_q)}{r!(t_{iq}-1-r)!} \left[D^r \frac{1}{w_q^{(t_i)}(s)} \right]_{s=z_q} \quad \textit{for } i = 1, \dots, n. \tag{8}$$

Proof It can be shown ([11, 6]) that

$$f_{[z_0(t_{i0}),\dots,z_{N-1}(t_{i,N-1})]} = \frac{1}{2\pi\imath} \oint_C \frac{f(s)}{\prod_{j=0}^{Q(i)-1}(s-z_j)^{t_{ij}}} ds$$

where C is a closed contour enclosing all points z_i [5, 6, 11]. From the Residue Theorem ([13])

$$f_{[z.t_i]} = \sum_{q=0}^{Q(i)-1} \frac{1}{(t_{iq}-1)!} \left[D^{t_{iq}-1} \frac{(s-z_q)^{t_{iq}} f(s)}{\prod_{l=0}^{N-1}(s-z_l)^{t_{il}}} \right]_{s=z_q}$$

with zeros being contributed to the sum whenever $t_{iq} = 0$. From Leibnitz's rule for the derivatives of a product

$$f_{[z.t_i]} = \sum_{q=0}^{Q(i)-1} \frac{1}{(t_{iq}-1)!} \sum_{r=0}^{t_{iq}-1} \binom{t_{iq}-1}{r} f^{(t_{iq}-1-r)}(z_q) \left[D^r \frac{1}{\prod_{\substack{l=0 \\ l \neq q}}^{N-1} (s-z_l)^{t_{il}}} \right]_{s=z_q}$$

which is the result as seen in Eq. (8). □

To remain consistent with the previous discussions and complexity counts we consider real z_is. It is however trivial to adapt the discussion for the complex field. In fact, all our results are equally valid for complex interpolation if we change the elementary operation unit to be defined over the complex field.

From Eq. (8) it also follows that the GDD can be viewed as a linear transformation on $\Re^n$

$$\delta = \mathbf{G}\phi \ , \tag{9}$$

where

$$\delta = (f_{[z.t_1]}, \dots, f_{[z.t_n]})^T,$$

$$\phi = (f_0, \dots, f_0^{(p_0-1)}, f_1, \dots, f_{N-1}, \dots, f_{N-1}^{(p_{N-1}-1)})^T.$$

Here $\mathbf{G}$ is the lower block triangular matrix

$$\begin{pmatrix} L_{00} & 0 & \cdots & \cdots & 0 \\ L_{10} & L_{11} & 0 & \cdots & 0 \\ L_{20} & \cdots & \cdots & \cdots & \cdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ L_{N-1,0} & L_{N-1,1} & \cdots & L_{N-2,N-1} & L_{N-1,N-1} \end{pmatrix}$$

in which $L_{ij} \in \Re^{p_i \times p_j}$ and the L_{ii} are lower triangular. In particular L_{00} is the diagonal matrix

$$L_{00} = \begin{pmatrix} 1 & 0 & . & . \\ 0 & \frac{1}{1!} & 0 & . \\ . & . & \ddots & . \\ . & . & . & \frac{1}{(p_0-1)!} \end{pmatrix} .$$

For example, when $p_i = 1$ for all i (the non-confluent case) each L_{ij} is reduced to a scalar and $\mathbf{G}$ is of order N. When $N = 1$, then $\mathbf{G}$ reduces to L_{00}.

It is central to this paper that the construction of the divided differences is reducible to the fast computation of Eq. (9). We distinguish two steps:

1. The computation of all elements of $\mathbf{G}$ (assembly phase).

2. The matrix-vector multiplication in Eq. (9).

We can already see that the time for step 2 is at most $O(\log n)$, or in terms of M and N, $O(\log N + \log M)$. In the subsequent sections we shall see how to obtain a fast algorithm for assembly step 1 and its combination with step 2.

4 Results and algorithm description

The arguments in this Section lead to a constructive proof of the main complexity result presented in Theorem 4.1.

For the moment let

$$\alpha_q^{(t_i)}(r; x) \equiv \left[D^r \frac{1}{w_q^{(t_i)}(s)} \right]_{s=x} \tag{10}$$

for any x, with

$$\alpha_q^{(t_i)}(0; x) = \frac{1}{w_q^{(t_i)}(x)} .$$

To motivate our discussion we examine in some detail the example started in Section 2. From Lemma 3.1 it follows that:

$$\begin{aligned} f_{[z.t_1]} &= f(z_0) \\ f_{[z.t_2]} &= \frac{f(z_0)}{1!0!} D\frac{1}{1} + \frac{f^{(1)}(z_0)}{0!1!}\frac{1}{1} \\ f_{[z.t_3]} &= \frac{f(z_0)}{1!0!}\left[D\frac{1}{s - z_1} \right]_{s=z_0} + \frac{f^{(1)}(z_0)}{0!1!}\frac{1}{z_0 - z_1} + \frac{f(z_1)}{0!0!}\frac{1}{(z_1 - z_0)^2} \\ &\cdots \end{aligned}$$

Using α as in Eq. (10) above, the matrix $\mathbf{G}$ in this case is

$$\left(\begin{array}{cc|c|ccc}
1 & 0 & \cdot & \cdot & \cdot & \cdot \\
0 & \frac{1}{1!} & 0 & \cdot & \cdot & \cdot \\
\hline
\frac{\alpha_0^{(t_3)}(1;z_0)}{1!0!} & \frac{\alpha_0^{(t_3)}(0;z_0)}{0!1!} & \alpha_1^{(t_3)}(0;z_1) & 0 & \cdot & \cdot \\
\hline
\frac{\alpha_0^{(t_4)}(1;z_0)}{1!0!} & \frac{\alpha_0^{(t_4)}(0;z_0)}{0!1!} & \alpha_1^{(t_4)}(0;z_1) & \alpha_2^{(t_4)}(0;z_2) & 0 & \cdot \\
\frac{\alpha_0^{(t_5)}(1;z_0)}{1!0!} & \frac{\alpha_0^{(t_5)}(0;z_0)}{0!1!} & \alpha_1^{(t_5)}(0;z_1) & \frac{\alpha_2^{(t_5)}(1;z_2)}{1!0!} & \frac{\alpha_2^{(t_5)}(0;z_2)}{0!1!} & 0 \\
\frac{\alpha_0^{(t_6)}(1;z_0)}{1!0!} & \frac{\alpha_0^{(t_6)}(0;z_0)}{0!1!} & \alpha_1^{(t_6)}(0;z_1) & \frac{\alpha_2^{(t_6)}(2;z_2)}{2!0!} & \frac{\alpha_2^{(t_6)}(1;z_2)}{1!1!} & \frac{\alpha_2^{(t_6)}(0;z_2)}{0!2!}
\end{array}\right)$$

The evaluation of $\mathbf{G}$ is centered around the evaluation of each of the blocks L_{qr}. Lemmas 4.1 and 4.2 demonstrate that for given q and i (i.e. in a given row of block L_{qr}), the terms $\alpha_q^{(t_i)}(r;z_q)$ satisfy a linear recurrence in r. To solve each of these recurrences, their coefficients and initial values must first be evaluated (Lemma 3.1 and Corollary 4.1). The recurrences are then solved as described in Lemma 4.3. Finally all of these steps are put together in the description of the algorithm in the proof of Theorem 4.1.

The proof of the following Lemma follows trivially after application of the rules of differentiation.

Lemma 4.1 *For $i = 1, \ldots, n$ and $q = 0, \ldots, N-1$, define*

$$\sigma_{q,j}^{(t_i)}(s) \equiv \sum_{\substack{l=0 \\ l \neq q}}^{N-1} \frac{t_{il}}{(s-z_l)^j} \quad , \tag{11}$$

where the sum is empty (and equal to 0) when $N = 1$. Then for $j \geq 1$

$$D\sigma_{q,j}^{(t_i)}(s) = (-j)\sigma_{q,j+1}^{(t_i)}(s) \quad .$$

From this Lemma it immediately follows that

$$D^\nu \sigma_{q,1}^{(t_i)}(s) = (-1)^\nu \nu! \sigma_{q,\nu+1}^{(t_i)}(s) \quad .$$

Lemma 4.2 *When $r \geq 1$*

$$\left[D^r \frac{1}{w_q^{(t_i)}(s)}\right]_{s=z_q} = \sum_{j=0}^{r-1} \binom{r-1}{j} (-1)^{j+1} j! \sigma_{q,j+1}^{(t_i)}(z_q) \left[D^{r-1-j} \frac{1}{w_q^{(t_i)}(s)}\right]_{s=z_q} \tag{12}$$

Proof Differentiating

$$D\left(\frac{1}{w_q^{(t_i)}(s)}\right) = -\frac{1}{w_q^{(t_i)}(s)} \sigma_{q,1}^{(t_i)}(s)$$

with σ defined as in Lemma 4.1, we see that the Lemma is valid for $r = 1$. What we have here is an expression for the derivative as a product of two known functions. Applying Leibnitz's theorem for the higher derivatives of a product of functions, it follows that for $r \geq 1$

$$\begin{aligned}
D^r\left(\frac{1}{w_q^{(t_i)}(s)}\right) &= -D^{r-1}\left(\frac{\sigma_{q,1}^{(t_i)}(s)}{w_q^{(t_i)}(s)}\right) \\
&= -\sum_{j=0}^{r-1} \binom{r-1}{j} D^j \sigma_{q,1}^{(t_i)}(s)\, D^{r-1-j}\left(\frac{1}{w_q^{(t_i)}(s)}\right),
\end{aligned}$$

and using Lemma 4.1

$$= \sum_{j=0}^{r-1} \left(\begin{array}{c} r-1 \\ j \end{array} \right) (-1)^{j+1} j! \sigma_{q,j+1}^{(t_i)}(s)\, D^{r-1-j}\left(\frac{1}{w_q^{(t_i)}(s)}\right) .$$

Evaluating at z_q the result follows. □

From Lemma 3.1, we seek an algorithm for the fast evaluation of Eq. (8) as i varies from 1 to n. By substituting the expression derived in Lemma 4.2 for the derivatives in Eq. (8), it seems that for the computation of each one of the GDD, a triple summation is required. However we next show that a combination of fast algorithms can be used to achieve a much more rapid evaluation.

From Lemma 4.2, for $r \geq 1$

$$\alpha_q^{(t_i)}(r; z_q) = \sum_{j=0}^{r-1} \lambda(j, q, r, t_i, z_q) \alpha_q^{(t_i)}(r-1-j; z_q) \tag{13}$$

where

$$\lambda(j, q, r, t_i, z_q) = (-1)^{j+1} \frac{(r-1)!}{(r-1-j)!} \sigma_{q,j+1}^{(t_i)}(z_q) \tag{14}$$

are the interaction coefficients.

Lemma 4.3 *Consider the array*

$$\begin{array}{ccccc}
\sigma_{0,j}^{(t_1)}(z_0) & \sigma_{1,j}^{(t_1)}(z_1) & \sigma_{2,j}^{(t_1)}(z_2) & . & \sigma_{N-1,j}^{(t_1)}(z_{N-1}) \\
\sigma_{0,j}^{(t_2)}(z_0) & \sigma_{1,j}^{(t_2)}(z_1) & \sigma_{2,j}^{(t_2)}(z_2) & . & \sigma_{N-1,j}^{(t_2)}(z_{N-1}) \\
\sigma_{0,j}^{(t_3)}(z_0) & . & . & . & \sigma_{N-1,j}^{(t_3)}(z_{N-1}) \\
. & . & . & . & . \\
\sigma_{0,j}^{(t_n)}(z_0) & \sigma_{1,j}^{(t_n)}(z_1) & \sigma_{2,j}^{(t_n)}(z_2) & . & \sigma_{N-1,j}^{(t_n)}(z_{N-1})
\end{array} \tag{15}$$

Each array element $\sigma_{q,j}^{(t_i)}(z_q)$ as defined in Eq. (11) represents the finite sequence

$$\{\sigma_{q,1}^{(t_i)}(z_q), \ldots, \sigma_{q,t_{iq}-1}^{(t_i)}(z_q)\}. \tag{16}$$

In particular the sequence is empty if $t_{iq} \leq 1$. Let M be the maximum of $\{p_0, \ldots, p_{N-1}\}$. Then all sequences defined as in Eq. (16) for each array element of Eq. (15) can be evaluated in $O(\log M + \log N)$ parallel operations.

Proof First observe that across the array in Eq. (15) each of the t_{il}'s takes all integer values from 0 to $t_{nl} \leq p_l \leq M$. Similarly j assumes values from 1 to $t_{iq} - 1$. From Eq. (11) the elements in Eq. (16) for every array entry are based upon linear combinations of terms

$$(z_q - z_l)^1, \ldots, (z_q - z_l)^{t_{nq}-1} . \tag{17}$$

The evaluation of all such terms can be achieved in $O(\log t_{nq})$ steps by means of the parallel prefix algorithm (Theorem A.1). By applying $N(N-1)$ concurrent instances of the same algorithm (for each q and each l) and noting that $t_{nq} = p_q$ the evaluation of all the terms in Eq. (17) can be done in less than $\lceil \log M \rceil$ parallel steps. The required divisions for Eq. (11) can be achieved in a single parallel step. Finally the additions require at most $O(\log N)$ steps. Hence the result follows. □

The first step in the evaluation of the GDDs from Eq. (8) consists of the calculation of all interaction coefficients $\lambda(j,q,r,t_i,z_q)$.

Corollary 4.1 *With M as defined above, the calculation of all*

$$\lambda(j,q,r,t_i,z_q) \ ; \quad 0 \le j \le r-1; \, 0 \le q \le N-1; \, 1 \le i \le n$$

for $r = 1,\ldots,t_{iq}-1$ can be carried out in $O(\log N + \log M)$ parallel steps.

Proof At first all of the differences

$$z_l - z_q \text{ for } 0 \le l \ne q \le N-1$$

are evaluated in a single parallel subtraction step (not contributing to the order of magnitude counts for the complexity of the algorithm). From Lemma 4.3 the calculation of all the σ terms in Eq. (14) above can be performed in $O(\log N + \log M)$ parallel steps. All the factorial coefficients can also be calculated in parallel in at most $O(\log M)$ steps by applying parallel prefix (Corollary A.1). Thus the result follows. □

We now proceed to the second major step of the algorithm.

Lemma 4.4 *Assuming all interaction terms λ are available, for each value of q and i, all of*

$$\alpha_q^{(t_i)}(r;z_q) \ ; \quad r = 0,\ldots,t_{iq}$$

can be calculated in $O(\log^2 t_{iq} + \log N)$ parallel steps.

Proof First we note that if $N = 1$, then $\mathbf{G}$ reduces to L_{00} which is a diagonal matrix consisting of terms $\frac{1}{k!}$ for $k = 0,\ldots,M-1$. From Corollary A.1 the evaluation can be completed in time $O(\log M)$. The key to the proof when $N > 1$ is the observation that Eq. (13) for each of the needed terms $\alpha_q^{(t_i)}(r;z_q)$ is a linear recurrence of order t_{iq}. Hence at first all initial values

$$\alpha_q^{(t_i)}(0;z_q) = \frac{1}{w_q^{(t_i)}(z_q)}$$

for $1 \le i \le n$ and $0 \le q \le N-1$ must be computed. The most complicated term here corresponds to $i = n$ with

$$\begin{aligned} w_q^{(t_n)}(z_q) &= (z_q - z_0)^{p_0} \ldots (z_q - z_{q-1})^{p_{q-1}} (z_q - z_{q+1})^{p_{q+1}} \ldots (z_q - z_{N-1})^{p_{N-1}} \\ &= (z_q - z_0)^{p_0 - 1}(z_q - z_0) \ldots (z_q - z_{q-1})^{p_{q-1}-1}(z_q - z_{q-1}) \cdot \\ &\quad\; (z_q - z_{q+1})^{p_{q+1}-1}(z_q - z_{q+1}) \ldots (z_q - z_{N-1})^{p_{N-1}-1}(z_q - z_{N-1}) \ . \end{aligned}$$

From Eq. (17) we have already available most partial products of the right hand side since $p_i - 1 = t_{ni} - 1$. In a single parallel step the partial products are completed by multiplying each $(z_q - z_k)^{p_k - 1}$ with $(z_q - z_k)$. This is done for all instances of q and t_i. The final products are then calculated in $O(\log N)$ steps by means of parallel prefix. After a parallel division step, all initial values $\alpha_q^{(t_i)}(0;z_q)$ are available. Finally, the order t_{iq} linear recurrences of Eq. (13) are solved for each of

$$\alpha_q^{(t_i)}(r;z_q) \text{ for } r = 1,\ldots,t_{iq}$$

and each fixed value of q and i. From Theorem A.2 this can be done in parallel time $O(\log^2 t_{iq})$, and the result follows. □

We have from Lemma 4.4 that all the recurrences, resulting as i and q take their possible values, can be solved concurrently in time at most $O(\log^2 M)$. The last two steps of the algorithm are described in the proof of the main Theorem.

Theorem 4.1 *All of the n generalized divided difference coefficients for the Hermite interpolating polynomial can be evaluated in $O(\log N + \log^2 M)$ steps when $M, N > 1$.*

Proof Using Lemma 3.1 we express each of the GDD as in Eq. (8). From Corollary 4.1 the coefficients of all recurrences in Eq. (13) can be evaluated in time $O(\log M + \log N)$. Next the initial values for each of the recurrences are calculated in $O(\log N)$ steps as in Lemma 4.4. With this information, the recurrences may be solved in time $O(\log^2 M)$. The next step evaluates the $\sum_{r=0}^{t_{iq}-1}$ summation of Eq. (8) since by now all the individual terms in the sum have been found. For fixed i and q this corresponds to a summation of t_{iq} terms which can be done in at most $\lceil \log t_i q \rceil$ parallel steps. Hence all sums can be evaluated concurrently in time at most $O(\log M)$. Finally all these independently calculated terms are added together using an additional $O(\log N)$ parallel steps. This procedure is applied for all n instances of the index i and all the GDD are obtained in this manner. Adding the times obtained above the result follows. $\square$

The next Corollary shows that the algorithm applied to the special cases of $M = 1$ (non-osculatory interpolation) or $N = 1$ is equivalent to computing the divided differences for the Newton form by means of the method in [8], or computing the coefficients of the truncated Taylor expansion around z_0 respectively.

Corollary 4.2 *1. When $M = 1$ the divided differences can be computed in $2\lceil \log N \rceil + 2$ parallel operations.*

2. When $N = 1$ the divided differences can be computed in $O(\log M)$ parallel operations.

Proof Let first $M = 1$. In this case $p_q = 1$ for all q, $n = N$ and

$$t_i = (\overbrace{1, \ldots, 1}^{i}, 0, \ldots, 0) \quad .$$

First all of the $z_q - z_l$ for $0 \leq q \neq l \leq N-1$ are calculated in a single parallel subtraction. Since all p_i's are equal to 1, the step described in Corollary 4.1 is empty. From Eq. (8), the formula for the divided differences becomes

$$f_{[z.t_i]} = \sum_{q=0}^{i-1} \frac{f(z_q)}{w_q^{(t_i)}(z_q)}$$

where

$$w_q^{(t_i)}(z_q) \quad = \quad \prod_{\substack{l=0 \\ l \neq q}}^{i-1} (z_q - z_l) \quad .$$

The step described in Lemma 4.4 is reduced to the calculation of the initial values

$$\alpha_q^{(t_i)}(0; z_q) = \frac{1}{w_q^{(t_i)}(z_q)} \quad .$$

As in the Lemma this is done as follows: parallel prefix is used for each q to calculate the products

$$\begin{array}{l}(z_q - z_0) \\ (z_q - z_0)(z_q - z_1) \\ \dots\dots\dots \\ (z_q - z_0)(z_q - z_1)\dots(z_q - z_{q-1})(z_q - z_{q+1})\dots(z_q - z_{N-1})\end{array}$$

in $\lceil \log(N-1) \rceil$ steps. After a parallel division step all of the

$$\alpha_q^{(t_i)}(0; z_q)$$

are obtained. Finally as described in Theorem 4.1 all the divided differences are computed by summing in parallel in at most $\lceil \log N \rceil$ steps. The total time for the algorithm is thus found to be exactly $\lceil \log(N-1) \rceil + \lceil \log N \rceil + 2$ which is at most $2\lceil \log N \rceil + 2$, agreeing with the time given in Theorem 2.1. This proves the first part of the Lemma. When $N = 1$ and M arbitrary, Eq. (8) reduces to

$$f_{[z.t_i]} = \frac{f_0^{(i-1)}}{(i-1)!} \quad .$$

Moreover $\mathbf{G}$ becomes the diagonal matrix L_{00}. Now all the required terms can be calculated in $\lceil \log M \rceil + 1$ parallel steps using Corollary A.1. Clearly, this is equivalent to taking the first M terms of the Taylor expansion for $f(z)$ around z_0. □

We remark that in in [8] it is shown that the presented algorithm for $M = 1$ is practical, in the sense that its numerical stability properties are similar to those of the serial algorithms.

5 Polynomial evaluation

As mentioned in [8], a fast algorithm for the interpolation would not be very useful unless an algorithm of comparable speed could be designed for the evaluation.

Theorem 5.1 *Given sufficiently many processors, a polynomial of degree $n-1$ written in its Newton representation can be evaluated in $2\lceil \log n \rceil + 2$ parallel arithmetic steps at points $\{s_1, \dots, s_k\}$.*

Proof Since we are not concerned with the exact number of processors, there can be arbitrarily many points of evaluation. The proof holds irrespective of whether some x_is are equal or not, and hence is a direct carry-over from [8]. First of all the values

$$s_l - x_j \ ; \quad i = 1, \dots, k \ ; \ \ j = 0, \dots, n-1$$

are evaluated in one parallel step. Using parallel prefix and an extra multiplication the evaluation of

$$\{\gamma_i \prod_{j=0}^{i-1} (s_l - x_j)\} \ ; \qquad i = 1, \dots, n-1$$

for $l = 1, \dots, k$ can be achieved in $\lceil \log(n-1) \rceil + 1$ steps. A parallel summation algorithm for each l (e.g. parallel prefix or binary tree) for the partial results completes the algorithm in an additional $\lceil \log n \rceil$ steps. □

6 Applications to Vandermonde systems

Using the combinatorial symbol

$$(N)_r = N(N-1)\cdots(N-r+1) \quad ,$$

the confluent Vandermonde matrix corresponding to the distinct points $\{z_q\}, q = 0, \ldots, N-1$ and the multiplicity vector $\mathbf{p}$ defined in Section 2 is of the form

$$U = (U_0|U_1|\cdots|U_{N-1}) \quad , \tag{18}$$

where each of the blocks $U_q \in \Re^{N \times p_q}$ and

$$U_q = \begin{pmatrix} 1 & 0 & 0 & . & 0 \\ z_q & 1 & 0 & . & 0 \\ z_q^2 & 2z_q & 2 & . & 0 \\ z_q^3 & 3z_q^2 & 6z_q & . & . \\ . & . & . & . & . \\ z_q^{N-1} & (N-1)_1 z_q^{N-2} & (N-1)_2 z_q^{N-3} & . & (N-1)_{p_q} z_q^{N-p_q-1} \end{pmatrix}$$

To solve the dual system

$$U^T a = b \tag{19}$$

a parallel solver could be applied directly. However as it happens with systems having a special structure (e.g. Toeplitz), lower complexity algorithms can be obtained. The connection with interpolation becomes clear after observing that the solution a of Eq. (19) is the vector of coefficients of the unique polynomial $p(z)$ such that $p^{(k)}(x_i) = \beta_i$, where β_i is the i^{th} element of b. As before, the points x_i come from the unrolling of the sequence z_q to include the repetitions. The use of divided differences is frequently recommended ([3, 9, 27]). The algorithm, whose sequential complexity is $O(n^2)$, proceeds in two distinct steps.

1. Compute the divided difference vector $c = [\gamma_0, \ldots, \gamma_{n-1}]^T$ corresponding to the interpolation information pairs $\{x_i, \beta_i\}, i = 0, \ldots, n-1$.
2. Transform the Newton form polynomial $\sum_{i=0}^{n-1} \gamma_i \prod_{j=0}^{i-1}(x - x_j)$ into power form $\sum_{i=0}^{n-1} \alpha_i x^i$. The uniqueness of polynomial interpolation implies $a = [\alpha_0, \ldots, \alpha_{n-1}]^T$.

Lemma 6.1 *Given the pairs $\{\gamma_i, x_i\}, i = 0, \ldots, n-1$ for the Newton polynomial representation*

$$P_{n-1}(x) = \sum_{i=0}^{n-1} \gamma_i \prod_{j=0}^{i-1} (x - x_j), \tag{20}$$

the coefficients $\{\alpha_i\}, i = 0, \ldots, n-1$ of the power form representation

$$P_{n-1}(x) = \sum_{i=0}^{n-1} \alpha_i x^i \tag{21}$$

can be computed in $O(\log n)$ parallel steps.

Proof (See also [17]). For $i = 0, \ldots, n-1$, the power form for the product $\prod_{j=0}^{i-1}(x - x_j)$ can be computed in parallel time $O(\log i)$ from Theorem A.3. One parallel step for the multiplication with the γ_i and a $O(\log i)$ parallel addition to group the coefficients corresponding to x^i returns the results in time $O(\log n)$. □

When $\mathbf{p} = (1, \ldots, 1)$, the matrix U in Eq. (18) becomes

$$U = \begin{pmatrix} 1 & 1 & . & . & 1 \\ z_0 & z_0^2 & z_0^3 & . & z_0^{N-1} \\ z_1 & z_1^2 & z_1^3 & . & z_1^{N-1} \\ z_2 & . & . & . & z_2^{N-1} \\ . & . & . & . & . \\ z_{N-1} & z_{N-1}^2 & z_{N-1}^3 & . & z_{N-1}^{N-1} \end{pmatrix}$$

In [12] a parallel algorithm of time complexity $O(N)$ is described for this case. Using the non-confluent ($M = 1$) version of the algorithm described in this paper (see also [8]) steps (1) and (2) can be completed in $O(\log N)$ parallel time. We summarize the discussion in the following Lemma.

Lemma 6.2 *The dual Vandermonde system in Eq. (19) can be solved using $O(\log^2 M + \log N)$ parallel operations.*

7 Conclusions

We have described algorithms for parallel interpolation, evaluation and some applications. This generalizes the work in [4, 8, 23]. The algorithms can be extended to handle the more general problem of Hermite-Birkhoff interpolation ([6, 25]), whenever well-poised.

It could be argued that an actual implementation may be impractical since by the time the size of the problem becomes large enough to justify the use of parallelism, polynomial interpolation could break down due to ill-conditioning. Certain point distributions however will delay this breakdown. The improvement in Lagrange interpolation when using Chebyshev rather than equidistant points is well known. Computing a good set of points is a challenging problem ([10, 16]) and we point to recent work in [22, 26] for Newton interpolation. We also point to [14] where an error analysis is performed of the divided difference based Vandermonde solver of Björck and Pereyra and its success is explained.

The algorithms presented here make heavy use of the parallel prefix algorithm, as well as of fast parallel algorithms for the solution of linear recurrences and polynomial multiplication. Even though the required number of processors is polynomial in the input size, the issue of exact processor count is left for future discussion. We only mention that if one is interested in the processor – time tradeoffs, there are many possibilities even in the simplest case of $M = 1$. This is mainly due to the variety of strategies one can follow for parallel prefix.

A Appendix

We review some known concepts and results which are used in the paper.

Let $*$ be an associative binary operation on a set T. The prefix computation problem is defined as follows: Given elements $y_1, \ldots, y_n \in T$ compute all n initial products (prefixes) $y_1 * y_2 \cdots * y_i$ for $i = 1, \ldots, n$. Parallel algorithms for this computation are called parallel prefix algorithms. The following result is well known and essential for the discussion ([19, 21]).

Theorem A.1 *The n input parallel prefix computation can be performed in $\lceil \log n \rceil$ parallel time.*

The next Corollary follows trivially from Theorem A.1,

Corollary A.1 *Given a positive integer n, all factorial terms*

$$1!, 2!, 3!, \ldots, n!$$

can be computed in $\lceil \log n \rceil$ parallel time.

The next results concern the parallel solution of lower triangular systems, or equivalently of linear recurrences ([24]) and fast polynomial multiplication ([23]).

Theorem A.2 *The triangular system of equations $Lx = f$, where L is a lower triangular matrix of order n, can be solved in $\frac{1}{2}\log^2 n + \frac{3}{2}\log n + 3$ parallel steps.*

Theorem A.3 *The coefficients of the power form representation of the product of m polynomials with real coefficients of degree $n-1$ each, can be computed in $O(\log mn)$ parallel steps.*

Proof From [23, Theorem 2.3]. □

References

[1] A. Aho, Hopcroft, J. E., and Ullman, J. D. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[2] Atwood, G. H. Parallel Lagrangian interpolation. In *Proc. 1988 Intl. Conf. Paral. Proc.* (Aug. 1988), D. H. Bailey, Ed., pp. 120–123.

[3] Björck, A., and Pereyra, V. Solution of Vandermonde systems of equations. *Math. Comp.* (1971), 893–903.

[4] Chin, F. Y. A generalized asymptotic upper bound on fast polynomial evaluation. *SIAM J. Comput. 5* (1976), 682–690.

[5] Davis, P. J. *Interpolation and Approximation.* Dover, 1975.

[6] Elsner, L., and Merz, G. Lineare Punktfunktionale und Hermite-Birkhoff-Interpolation. *Beiträge zur Numerischen Mathematik 4* (1975), 69–82.

[7] Eğecioğlu, O., Gallopoulos, E., and Koç, Ç. *Parallel Hermite interpolation: An algebraic approach.* Tech. Rep. 671, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, July 1987, to appear in *Computing.*

[8] Eğecioğlu, O., Gallopoulos, E., and Koç, Ç. *Fast and practical parallel polynomial interpolation.* Tech. Rep. 646, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, January 1987.

[9] Galimberti, G., and Pereyra, V. Solving confluent Vandermonde systems of Hermite type. *Numer. Math. 18* (1971), 44–60.

[10] Gautschi, W. Optimally conditioned Vandermonde matrices. *Numer. Math. 24* (1975), 1–12.

[11] Gel'fond, A. O. *Calculus of Finite Differences.* Hindustan Publishers, 1971.

[12] Gohberg, I., Kailath, T., Koltracht, I., and Lancaster, P. Linear complexity parallel algorithms for linear systems of equations with recursive structure. *Linear Algebra and its Applications 88/89* (April 1987), 271–315.

[13] Henrici, P. *Applied and Computational Complex Analysis.* Vol. 1, Wiley, 1974.

[14] Higham, N. J. *Error analysis of the Björck-Pereyra algorithms for solving Vandermonde systems.* Tech. Rep. Numer. Anal. 108, Department of Mathematics, University of Manchester, December 1985.

[15] Horowitz, E. A fast method for interpolation using preconditioning. *IFIP Letters* (1972), 157–163.

[16] Kilgore, T. A. A characterization of the Lagrange interpolating projection with minimal Chebyshev norm. *J. Approx. Theory 24* (1978), 273–288.

[17] Koç, Ç. *Parallel algorithms for interpolation and approximation.* PhD thesis, Dept. Elec. Comput. Eng., University of California Santa Barbara, June 1988.

[18] Krogh, F. Efficient algorithms for polynomial interpolation and divided differences. *Math. Comp. 24* (January 1970), 185–190.

[19] Kruskal, C. P., Rudolph, L., and Snir, M. The power of parallel prefix. *IEEE Trans. Comput. C-34*, 10 (October 1985), 965–968.

[20] Kung, H. T. *Fast evaluation and interpolation.* Tech. Rep., Department of Computer Science, Carnegie-Mellon University, 1973.

[21] Ladner, R., and Fischer, M. Parallel prefix computation. *J. Assoc. Comput. Mach. 27* (1980), 831–838.

[22] Reichel, L. *Newton interpolation in Fejér and Chebyshev points.* Tech. Rep. 88/24, IBM Bergen Scientific Centre, May 1988.

[23] Reif, J. Logarithmic depth circuits for algebraic functions. *SIAM J. Comput. 15* (1986), 231–242.

[24] Sameh, A. H., and Brent, R. Solving triangular systems on a parallel computer. *SIAM J. Numer. Anal. 14* (1977), 1101–1113.

[25] Sharma, A. Some poised and nonpoised problems of interpolation. *SIAM Rev. 14*, 1 (January 1972), 129–151.

[26] Tal-Ezer, H. *High degree interpolation polynomial in Newton form.* Tech. Rep. 88-39, ICASE, 1988.

[27] Tang, W. P., and Golub, G. H. The block decomposition of a Vandermonde matrix and its applications. *BIT 21* (1981), 505–517.

[28] Traub, J. F. *Iterative Methods for the Solution of Equations.* Prentice-Hall, Englewood Cliffs, N.J., 1964.

[29] Tsao, N. K., and Prior, R. On multipoint numerical interpolation. *ACM Trans. Math. Softw. 4*, 1 (March 1978), 51–56.