Parallel Computing 18 (1992) 75–83
North-Holland

Systolic algorithm for rational interpolation and Padé approximation

V.K. Murthy [a], E.V. Krishnamurthy [b], and Pin Chen [b]

[a] *Fujitsu Australia Ltd., Sydney, Australia*
[b] *Computer Science Laboratory, Research School of Physical Science and Engineering, The Australian National University, GPO Box 4, ACT 2601, Canberra, Australia*

Received June 1990
Revised August 1990

Abstract

Murthy V.K., E.V. Krishnamurthy and P. Chen, Systolic algorithm for rational interpolation and Padé approximation, Parallel Computing 18 (1992) 75–83.

This paper describes a systolic algorithm for rational interpolation based on Thiele's reciprocal differences. This algorithm constructs the continued fraction/Padé approximant of a stream of input data using a linear array of processors. The period of this algorithm is $O(n+1)$ (where $n+1$ is the number of distinct points at which the function values are available) to produce an M/M Padé approximant ($M = n+1/2$, n odd; $M = n/2$, n even) using $n+1$ processors. For illustrative purpose the Connection Machine implementation of this systolic algorithm in CM Fortran is presented with an example.

Keywords. Systolic algorithm; rational interpolation; Padé approximation; Connection Machine.

1. Introduction

In a recent paper [3], we described a systolic algorithm for polynomial interpolation and Chinese remaindering based on Newton's divided differences scheme. This paper is an extension of this concept for rational interpolation based on Thiele's reciprocal differences and continued fraction approximation, Baker and Grave-Morris [1]. The rational interpolation which generates a ratio of two polynomials constitute a richer class of functions than the ordinary polynomials. Further, the rational Padé approximant (M/N) whose numerator is of degree M and denominator is of degree N has the same accuracy as a polynomial of degree $M+N$, see Cheney [2]; besides, the rational interpolation provides for rounding and representational errors and for the detection of singularities.

Further, a rational function when available as a continued fraction is more economically evaluated with at most $\max(M, N)$ multiplications or division operations; this is also true when numerator and denominators are evaluated independently in parallel [2]. This algorithm is implemented on the Connection Machine with a linear array of processors, using CM Fortran.

2. Rational interpolant

The algorithm to be described here yields a continued fraction representation of the $(n+1)$ point Padé approximation that fits the function values at $(n+1)$ distinct points. This algorithm can be carried out using any arithmetic system; what are crucial here are the time synchronization, local communication and a clock control. Hence, this algorithm can be implemented using systolic arrays, or Instruction Systolic Arrays (ISA) or can be programmed in Connection Machine or vector array processors. Here we will describe the CM implementation.

Let a function $r(x)$ be specified at $(n+1)$ points x_i such that

$$r(x_i)=r_i \quad \text{for } i=0,1,\dots,n.$$

We need to determine $r(x)$ as a rational polynomial over a specified field.

Let the list $\{X_k\}$ denote the set $\{x_0, x_1,\dots,x_k\}$ and the list $\{X_k, x\}$ denote $\{x_0, x_1,\dots,x_k, x\}$ which is obtained by appending to $\{X_k\}$ the element x. We denote $P\{X_0\}=P(x_0)=r_0=P_0$ and $P\{X_i\}=P_i$.

We then define

$$\begin{aligned} P\{X_1\} &= P_1 = P\{x_0, x_1\} = (x_1-x_0)/(r_1-r_0)\\ P\{X_2\} &= P_2 = P\{X_1, x_2\}\\ &= [(x_2-x_1)/(P\{X_0, x_2\}-P\{X_1\})] + P\{X_0\}, \end{aligned}$$

and in general, for a specified i,

$$P\{X_i, x_{i+j}\} = [(x_{i+j}-x_i)/(P\{X_{i-1}, x_{i+j}\}-P\{X_i\})] + P\{X_{i-1}\}$$

for all $j=1,2,\dots,(n-1)$.

Note that the above recursion for rational interpolation is different from that for polynomial interpolation described in [3].

The interpolant $r(x)$ on $x_0, x_1,\dots,x_n$ is then obtained as the following continued fraction, Baker et al. [1]:

$$r(x)=P_0+\cfrac{x-x_0}{P_1+\cfrac{x-x_1}{P_2-P_0+\cfrac{x-x_2}{P_3-P_1+\dots+\cfrac{x-x_{n-1}}{P_n-P_{n-2}}}}}$$

which will be also denoted by

$$r(x)=P_0+\frac{x-x_0}{P_1+}\,\frac{x-x_1}{P_2-P_0+}\,\frac{x-x_2}{P_3-P_1+}\,\frac{x-x_{n-1}}{P_n-P_{n-2}}.$$

This construction of the Padé approximant can be expressed as the following recurrences:

$$\begin{aligned} &A_0=P_0;\ B_0=1;\\ &A_1=P_1P_0+(x-x_0);\ B_1=P_1;\\ &A_{i+1}=A_i(P_{i+1}-P_{i-1})+A_{i-1}(x-x_i) \quad \text{for } i=1,2,\dots,n-1\\ &B_{i+1}=B_i(P_{i+1}-P_{i-1})+B_{i-1}(x-x_i) \quad \text{for } i=1,2,\dots,n-1\\ &r=A_n/B_n. \end{aligned}$$

The above recurrences to compute A_i and B_i $(i=1,2,\dots,n-1)$ can be expressed as the product of a (2×2) matrix and a (2×1) vector:

$$\begin{bmatrix} A_{i+1}\\ B_{i+1}\end{bmatrix} = \begin{bmatrix} A_i & A_{i-1}\\ B_i & B_{i-1}\end{bmatrix}\begin{bmatrix} P_{i+1}-P_{i-1}\\ x-x_i\end{bmatrix}.$$

Note that the resulting vector is fed back in the next iteration as the left column of the right-side matrix; also, the previous left column moves to the right column. This permits a synchronous computation in a systolic machine such as a Connection Machine.

Thus, we obtain the $(n+1)$ point Padé approximant $[X/Y]$ where X denotes the degree of the numerator and Y the degree of the denominator.

If $n+1=2M+1$ (odd) where M is an integer then we get an $[M/M]$ approximant. This means that we need to determine $M+1$ coefficients in the numerator and M coefficients in the denominator (the constant term in the denominator is taken as unity).

Otherwise, if $n+1=2M+2$ (even) we get an $[(M+1)/M]$ approximant. This means that we need to determine $M+2$ coefficients in the numerator and M coefficients in the denominator.

For example, if $n+1=6$, let the values available at points x_i be r_i $(i=0, 1, \ldots, 5)$. We can then successively construct the [0/0], [1/0], [1/1], [2/1], [2/2], [3/2] Padé approximants, if they exist.

If some intermediate step involves a division by a zero, then the Padé approximant does not exist; we will explain later how to deal with this problem.

We now describe the parallel algorithm for rational interpolation.

3. Algorithm

This algorithm takes as inputs (x_i, r_i) $(i=0, 1, \ldots, n)$ such that $\{r_i\}$ are the values of the function at points $\{x_i\}$. The algorithm reconstructs a Pade approximant $r=A_n/B_n$ of order $[((n+1)/2)/((n-1)//2)]$ when n is odd and $[(n/2)/(n/2)]$ when n is even.

We assume that there are $n+1$ processors PRO(i) $(i=0, 1, \ldots, n)$. The registers in each processor are indexed by i.

Input: (x_i, r_i) $(i=0, 1, \ldots, n)$.

Register R_i is initialized with r_i and register X_i is initialized with x_i.

Output: $r=A_n/B_n$

The algorithm is given below:

```
$P_{-1} := 0$;
**For** $i = 0$ to $n$ **do**
**begin**
$R_i := r_i$;
$X_i := x_i$;
**end**;
**For** $j = 0$ to $n$ **do**
**begin**
$P_j := R_j$;
**For** $s = j+1$ to $n$ in parallel **do**
**begin**
$S_s := R_j$;
$R_s := R_s - S_s$;
**If** $R_s = 0$ **then** 'ABORT';
**else**
$M_s := X_s - X_j$;
$R_s := M_s/R_s$;
$R_s := R_s + P_{j-1}$;
**end**;
**end**;
```

The Padé Approximant can then be constructed as follows:

```
$A_0 := P_0$; $B_0 := 1$;
$A_1 := P_1P_0 + (x - X_0)$; $B_1 := P_1$;
for $i = 1$ to $n - 1$ do
begin
$A_{i+1} := A_i(P_{i+1} - P_{i-1}) + A_{i-1}(x - X_i)$;
$B_{i+1} := B_i(P_{i+1} - P_{i-1}) + B_{i-1}(x - X_i)$;
end;
$r := A_n/B_n$;
```

Remarks

1. In the last iteration of j, when j is n, 'for $s = j + 1$ to n in parallel do begin' is not executed since $j + 1$ equals $n + 1$, and exceeds the limit n.

2. The algorithm aborts when a division by zero arises.

Let $r(-1) = r_0 = 1$, $r_0 = r_1 = 1$, Then the algorithm aborts when $(r_1 - P_0) = 0$ $(P_0 = r_0)$. Hence there is no rational function of type [1/1] that fits this data.

4. Connection Machine implementation

In theory, if this algorithm were implemented with sequential programming, the complexity of the algorithm should be $(O(n + 1)^3)$. Taking advantage of the data-level parallelism of Connection Machine, this algorithm speeds up to $O(n + 1)$. For illustrative purpose and for providing a clear understanding, the core of the program in CM Fortran is presented below:

```
**********Parallel Rational Interpolation**********
        double precision x(n + 2), r(n + 2), p(n + 2), s(n + 2), m(n + 2), a1(n + 2)
        double precision a2(n + 2), b2(n + 2), c1(n + 2), c2(n + 2), b1(n + 2)
        integer i, h, j
CMF$    layout x(:news), r(:news), p(:news), s(:news), a2(:news), c2(:news)
CMF$    layout m(:news), a1(:news), b1(:news), b2(:news), c1(:news)
        :
        p = 0.0
        do j = 2, n + 1
          p(j) = r(j)
          i = j + 1
          s(i:n + 2) = r(j)
          if (count(r(i:n + 2). eq. s(i:n + 2)) ≥ 1) then subroutine abort
          r(i:n + 2) = r(i:n + 2) - s(i:n + 2)
          m(i:n + 2) = x(i:n + 2) - x(j)
          r(i:n + 2) = m(i:n + 2)/r(i;n + 2)
          r(i:n + 2) = r(i:n + 2) + p(j - 1)
        enddo
        p(n + 2) = r(n + 2)
        :
        stop
        end
```

This implementation was achieved with parallelism and efficient interprocessor communication dynamically using $n + 1$ processor. Each processor computes one of $\{P_i\}$ independently.

i	x_i	y_i	z_i
0	2	17/9	9/17
1	3	41/14	14/41
2	4	257/65	65/257
3	5	313/63	63/313
4	6	1297/217	217/1297
5	7	1251/172	172/1251
6	8	4097/513	513/4097
7	9	3281/365	365/3281

Fig. 1. Function values of y and z at points $n+1=8$.

Actually, for any processor Pro(i) ($i = 2, 3, \ldots, n+2$), the period of execution is O($i+1$). Only Pro (n) takes O($n+1$) because of the systolic property of the algorithm.

5. Computational result

The above program was run on the Connection Machine with $n+1$ processors corresponding distinct $n+1$ points at which the function values are available so that their rational interpolation and Padé approximations can be achieved.

For example, consider the functions specified by *Fig. 1* where $y = (x^4+1)/(x^3+1)$ and its reciprocal $z = (x^3+1)/(x^4+1)$.

Using our algorithm, we can get two vectors $P1$ and $P2$ respectively corresponding function values of y and z.

$$P1 = \{1.8888889, 0.96183206, 150.89830, 1.0428083, 93.326530, 1.0005695, -17395.000, 1.0000000\}$$

$$P2 = \{0.52941176, -5.320611, 6.6269797, -2000.0715, 1.0715067, -1725.504, -5.7487784, 187985.00\}$$

Here, we assume a polynomial function $f(x) = (a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4)/(1 + b_1x + b_2x^2 + b_3x^3 + b_4x^4)$. Using $P1$ and $P2$, the Padé approximants for $\{y_i\}$ And $\{z_i\}$ are constructed as shown in *Figs. 2* and *3*.

Incidentally, as one expects, the Padé approximants for y are the reciprocal of that for z with their numerator degrees and denominator degrees being equal. This observation provides a method to interpolate a function having singularities at some points. In such case, we can use the inverse function for interpolation if it has no further sigularities at other points. For example at $x = -1$ the function $y(-1) = \infty$, we may then attempt to interpolate its inverse function by using $z(-1)$, and obtain its reciprocal Padé approximants.

$\frac{D_n}{D_d}\{y\}$	a_0/b_0	a_1/b_1	a_2/b_2	a_3/b_3	a_4/b_4
0/0	1.8888889				
	1				
1/0	−0.1832061	−0.1904762			
	1				
1/1	−0.2349318	−1.0753714			
	1	0.0071265			
2/1	−0.2576897	1.06926401	0.1330599		
	1	0.1387560			
2/2	−0.2436128	1.1021051	−0.0822778		
	1	−0.0663728	−0.0008816		
3/2	−0.0183876	0.9403920	−0.6762982	0.3548192	
	1	−0.6817539	0.3550212		
3/3	0.0248621	0.9026555	−0.7601335	0.4206975	
	1	−0.7724920	0.4215440	−0.0000241	
4/3	1	0	0	0	1
	1	0	0	1	

Fig. 2. Padé approximant coefficients for y.

In Section 2 we have mentioned that a division by a zero may abort the algorithm. Since all processors compute independently and some of the computational results from the first i points have been used by ith loop, a practical way to avoid abort is to remove that point at which a division by a zero arises; then continue to interpolate with the remaining points. This modification can be introduced in the earlier algorithm, as described below. Such a modification, however, will result in a lower order Padé approximant.

```
:
For $s = j + 1$ to $n$ in parallel do
begin
$S_s := R_j$;
if $R_s = S_s$ then subroutine abort
begin
```

$\frac{D_n}{D_d}\{z\}$	a_0/b_0	a_1/b_1	a_2/b_2	a_3/b_3	a_4/b_4
0/0	0.5294118				
	1				
1/0	−4.81679	0.9053085			
	1				
1/1	−4.2565553	−0.0303342			
	1	−4.5773779			
2/1	−4.0162029	−0.048212	0.0021876		
	1	−4.3758818			
2/2	−4.1048744	0.2724520	0.0036189		
	1	−4.5240021	0.3377400		
3/2	−3.9788422	0.7680777	0.0130058	−0.0005183	
	1	−4.5503624	0.8942987		
3/3	40.22179	−31.071012	16.955253	−0.0009728	
	1	36.306420	−30.57393	16.921206	
4/3	14.618043	−12.339738	7.0856382	−0.0016512	0.00003753
	1	12.567230	−12.016099	7.0546403	

Fig. 3. Padé approximant coefficients for z.

```
For $l = s$ to $n - 1$ in parallel do
$X_l := X_{l+1}$;
$R_l := R_{l+1}$;
$r_l := r_{l+1}$;
$S_l := S_{l+1}$;
end
$R_s = R_s - S_s$;
$M_s := X_s - X_j$;
 .
 .
 .
```

The result of our experiments shows that the rational Padé approximations we get using this algorithm provide very good approximations to the known original functions which are provided in the Padé table [4].

$f(x)$	a_0/b_0	a_1/b_1	a_2/b_2	a_3/b_3	a_4/b_4	a_5/b_5
e^x	0.9760343	0.7638769	0.1751546	0.0813596	−0.0008916	0.0019788
	1	−0.2979991	0.0346652	−0.0018611	0.0000388	
$\ln(x)$	−2.990106	−2.2925167	3.9111002	1.3120769	0.0593472	0.0000988
	1	4.8711509	3.3400615	0.4821936	0.0124550	
$\mathrm{Sin}(x)$	−0.2179286	1.5800787	−0.9814790	0.2134807	−0.019114	0.0005998
	1	−0.3854562	0.0766791	−0.0074469	0.0003087	
$\frac{x^3+1}{x^4+1}$	1	0	0	1	0	0
	1	0	0	0	1	

Fig. 4. Padé table.

Also in *Fig. 4* we provide rational interpolations and Padé approximation table for some special functions where $\{x_i\} = \{1, 2, 3, 4, \ldots, 10\}$.

$$e^x = 2.7182818 + \frac{x-1}{0.21409727+}\,\frac{x-2}{-10.107338+}\,\frac{x-3}{-0.37162123+}\,\frac{x-4}{27.474593+}$$

$$\frac{x-5}{0.24444871+}\,\frac{x-6}{-74.683687+}\,\frac{x-7}{-0.12956184+}\,\frac{x-8}{203.01131+}\,\frac{x-9}{0.06224374}$$

$$\ln x = 0.0 + \frac{x-1}{1.4426950}\,\frac{x-2}{2.6470193+}\,\frac{x-3}{7.9637249+}\,\frac{x-4}{1.3862190+}\,\frac{x-5}{19.862082+}$$

$$\frac{x-6}{0.94355679+}\,\frac{x-7}{37.112163+}\,\frac{x-8}{0.71617245+}\,\frac{x-9}{59.706509}$$

$$\sin(x) = 0.84147098 + \frac{x-1}{14.743513+}\,\frac{x-2}{-0.0568207+}\,\frac{x-3}{-15.744967+}$$

$$\frac{x-4}{-1.0868184+}\,\frac{x-5}{1.6824286+}\,\frac{x-6}{4.1982829+}\,\frac{x-7}{-0.56362272+}$$

$$\frac{x-8}{-14.505402+}\,\frac{x-9}{0.39733903}$$

$$\frac{x^3+1}{x^4+1} = 1 + \frac{x-1}{-2.125}\,\frac{x-2}{-1.0964467+}\,\frac{x-3}{25.909783+}\,\frac{x-4}{0.10197254+}\,\frac{x-5}{-1199.3865+}$$

$$\frac{x-6}{-0.00484652+}\,\frac{x-7}{50000.602+}\,\frac{x-8}{0.0001.9596+}\,\frac{x-9}{3.424E+13}$$

Concluding remarks

The above rational interpolation scheme can be used in conjunction with fast evaluation and matrix inversion schemes in the Connection Machine, to invert rational polynomial matrices. Since the inversion of polynomial matrices are computationally intensive, data parallelism can speed up these computations. These applications will be reported elsewhere.

References

[1] G.A. Baker and P. Grave-Morris, *Padé Approximants* (vols. 1 and 2) (Addison-Wesley, Reading, MA, 1981).
[2] E.W. Cheney, *Introduction to Approximation Theory* (McGraw Hill, New York, 1966).
[3] H. Schroder V.K. Murthy, and E.V. Krishnamurthy, Systolic algorithm for polynomial interpolation and related problems, *Parallel Comput.* 17 (4 & 5), (1991) 493-503.
[4] D.C. Handscomb et al., *Methods of Numerical Approximation* (Pergamon, Oxford, 1966).