Parallel Computing 17 (1991) 493–503
North-Holland

Systolic algorithm for polynomial interpolation and related problems

H. Schroder [a], V.K. Murthy [b] and E.V. Krishnamurthy [c]

[a] *Department of Electrical Engineering and Computer Science, University of Newcastle, Newcastle, N.S.W. 2308, Australia*
[b] *Fujitsu Australia Ltd., Sydney, Australia*
[c] *Computer Science Laboratory, Research School of Physical Sciences, Australian National University, Canberra, ACT 2601, Australia*

Received August 1989
Revised June 1990

Abstract

Schroder, H., V.K. Murthy and E.V. Krishnamurthy, Systolic algorithm for polynomial interpolation and related problems, Parallel Computing 17 (1991) 493–503.

This paper describes a systolic algorithm for interpolation and evaluation of polynomials over any field using a linear array of processors. The periods of these algorithms are O(n) for interpolatin and O(1) for evaluation. This algorithm is readily adapted for Chinese remaindering, easily generalized for the multivariable interpolation and can be extended for rational interpolation to produce Pade approximants. The instruction systolic array implementation of the algorithm is presented here.

Keywords. Chinese remaindering; evaluation; interpolation; polynomials; instruction systolic arrays; Lagrange interpolant; Newton interpolation; period of algorithm; quorum security locks; systolic algorithms; systolic architectures.

1. Introduction

Systolic algorithms are highly suitable for implementation in a regular network of processors, where each processor has a simple instruction set and has a provision for local communication among the neighbouring processors. In this paper, we describe how the classical polynomial interpolation/evaluation over any field and the related Chinese remaindering problem can be solved using the systolic approach.

The conventional systolic array architecture (SA) lacks flexibility to execute a class of abstract algorithms in which the inputs and outputs belong to different data domains. To improve this situation the concept of Instruction Systolic Arrays (ISA) due to Lang [3] is very useful; the ISA retains all the advantages of systolic architectures yet permits the execution of different closely related algorithms defined under different data domains using the same processor array.

In the ISA, a sequence of instructions called the Top Program (TP), and an orthogonal sequence of Boolean selectors called the Left Program (LP) are pumped through the mesh connected array of processors (unlike SA, where only the data is pumped through the processors). If an instruction meets the selector bit '1' at the processor, then the instruction is executed in that processor; otherwise, (i.e. it meets a '0' bit) the instruction is not executed in that processor leaving the contents of its registers unchanged. Thus an ISA program is a 2-tuple (TP, LP); data is supplied to ISA from external data queues, which can be read during the execution. Hence one can execute different programs on the same architecture simply by pumping in different set of instructions; also the flexibility of ISA is enhanced by the ability to inhibit instructions at certain processors using selectors. These two features make ISA a programmable systolic architecture.

Also in contrast to the mesh-connected processor array which consists of independent processors each with its own memory and program storage, the ISA processors have no local memory for program storage; hence they can be realised in a smaller chip area.

2. Polynomial interpolation / Chinese remaindering

The polynomial interpolation over a field (real or finite) consists in finding the $(n+1)$ coefficients of an nth degree polynomial in the specified field, given the functional values at $(n+1)$ distinct points; obviously the values of the polynomial evaluated at these points should coincide. A closely related problem is the computation of the solution to a system of linear congruences over the integers based on the Chinese remainder theorem, see Krishnamurthy [1]. Both these algorithms have tremendous applications in coding theory, design of quorum security locks, see Shamir [7], McEliece and Sarvate [4] and exact computation [1].

From an algebraic point of view the Chinese remaindering and polynomial interpolation problems are equivalent. Given a set of remainders (residues) $\{r_0, r_1, \ldots, r_n\}$ with respect to a set of moduli $\{p_0, p_1, \ldots, p_n\}$ which are pairwise relatively prime, both problems reconstruct an element r such that $r_i = r \bmod p_i$ for $i = 0, 1, \ldots, n$ in respective domains; r is uniquely determined if

$$\text{size } r < \text{size} \prod_{i=0}^{n} p_i = M,$$

where size is suitably defined as follows: for integers 'the size' is the magnitude and for polynomials 'the size' is the degree. Element r is defined as

$$r = \sum_{i=0}^{n} (M/p_i) r_i T_i \bmod M,$$

where T_i is the solution $(M/p_i)T_i = 1 \bmod p_i$.

The equivalence between the Chinese remaindering algorithm and the Lagrange polynomial interpolation is readily seen from the following correspondence [1]:

$$r(x) = n\text{th degree polynomial in a field } F$$

$$r(x_i) = r_i \quad (i = 0, 1, \ldots, n)$$

$$p_i(x) = (x - x_i)$$

$$r(x) \bmod p_i(x) = r_i$$

$$M = \prod_{i=0}^{n} p_i(x).$$

The Lagrange interpolant is obtained from

$$L_n(x) = \sum_{i=0}^{n} r_i T_i \prod_{k=0}^{n} (x - x_k),\ k \neq i,$$

where

$$T_i = \frac{1}{\prod_{k=0}^{n} (x_i - x_k)} = [M/p_i(x)]^{-1} \bmod (x - x_i)$$

(M and $p_i(x)$ are defined above), $k \neq i$.

Since in doing polynomial interpolation, we restrict ourselves to linear polynomials $(x - x_i)$ $(i = 0, \ldots, n)$, the remainder or residue computation for a polynomial $r(x)$ is equivalent to

$r(x) \bmod (x - x_i) = r_i$ by the remainder theorem.

Thus, the reconstruction of an integer r in the range $0 \leqslant r \leqslant M - 1$ from a set of residues $\{r_0, r_1, \ldots, r_n\}$ with respect to a set of primes $\{p_0, p_1, \ldots, p_n\}$ and the reconstruction of a polynomial $r(x)$ of degree atmost n from a set of residues $\{r_0, r_1, \ldots, r_n\}$ with respect to a set of distinct linear polynomials $\{p_0(x), p_1(x), \ldots, p_n(x)\}$ can be achieved by identical algorithms for suitably defined data domains.

We now formally state the sequential algorithm.

2.1. Sequential algorithm

The sequential algorithm given below takes as inputs residues $\{r_0, r_1, \ldots, r_n\}$ and moduli $\{p_0, p_1, \ldots, p_n\}$. For integers, r is reconstructed such that $r \bmod p_i = r_i$ where r is over modulo M, $M = \prod_{i=0}^{n} p_i$. For polynomials, $r(x)$ is reconstructed such that $r(x) \bmod p_i = r_i$ where $p_i = x - x_i$; here, $r(x)$ is a nth degree polynomial over field F.

```
Input: residues r_i and moduli p_i (i=0, 1,..., n).
Output: reconstructed element r
 M:=1;
 R:=r_0;
 for k=1 to n do
 begin
  M:=Mp_{k-1};
  u:=M^-1 mod p_k;
  d:=(r_k-R)u mod p_k;
  R:=R+dM
 end;
r:=R;
```

This algorithm expresses r in the form $r = d_0 + d_1 p_0 + \cdots + d_n P_0 \cdots p_{n-1}$, where $d_0 = r_0$. For integers, r is in mixed-radix form where d_i's are the mixed-radix digits and $0 \leqslant d_i \leqslant p_i - 1 (i = 0, 1, \ldots, n)$. For polynomials, $r(x) = d_0 + d_1(x - x_0) + \cdots + d_n(x - x_0) \ldots (x - x_{n-1})$ is in the Newton's interpolation form, where $p_i = (x - x_i)$. We now explain how the above algorithm can be converted to a parallel/distributed form.

3. Parallel/distributed algorithn for Chinese remaindering/interpolation

In the description of this parallel algorithm, we do not constrain ourselves to a particular parallel computational model. Suitable modifications, however, can be easily incorporated to suit a particular computational model. Its ISA implementation will be desribed later.

This algorithm takes as inputs residues $\{r_0, r_1, \ldots, r_n\}$ and moduli $\{p_0, p_1, \ldots, p_n\}$. For integers, r is reconstructed such that $r \bmod p_i = r_i$ where r is over modulo M,

$$M = \prod_{i=0}^{n} p_i.$$

For polynomials, $r(x)$ is reconstructed such that $r(x) \bmod p_i = r_i$ where $p_i(x) = x - x_i$; here, $r(x)$ is an nth-degree polynomial over field F.

We assume that there are $n+1$ processors PRO(i) ($i = 0, 1, \ldots, n$), each with five registers R_i, D_i, S_i, P_i and M_i. The operations subtraction, multiplication and inversion are carried out in the appropriate field for each processor.

Register R_i is initialized with r_i and register P_i with p_i (for polynomials, register X_i is initialized with x_i). At the end of the algorithm, register D_i in each processor contains the coefficient d_i, where $d_0 = r_0$. We can then reconstruct r using D_i.

```
Input: residues r_i and moduli p_i(i=0, 1,..., n).
Output: reconstructed element r
For i=0 to n do
begin
 R_i:=r_i;
 P_i:=p_i;
end;
For j=0 to n do
begin
 D_j:=R_j;
 For s=j+1 to n in parallel do
 begin
  S_s:=R_j;
  R_s:=R_s-S_s;
  M_s:=P_j^-1 mod P_s;
  R_s:=R_sM_s mod P_s;
 end;
end;
```

Then r: $= D_0 + \sum_{j=1}^{n} D_j \prod_{s=0}^{j-1} P_s$.

In the case of polynomials, we have $P_j = (x - X_j)$, $P_s = (x - X_s)$. The last three assignment statements in the above algorithm then are:

$$M_s := (X_s - X_j)^{-1};$$

$$R_s := R_s M_s;$$

$$r = D_0 + \sum_{j=1}^{n} D_j \prod_{s=0}^{j-1} x - X_s.$$

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. Computing r using the above formula can be deferred until one wants to evaluate the polynomial at some point.

3.1. Proof of algorithm

We now provide a proof of correctness of the algorithm.

3.1.1. Chinese remaindering

In order to reconstruct the integer r we assume that it is expressed uniquely in the mixed-radix form

$$r = d_0 + d_1 p_0 + \cdots + d_n P_0 \dots P_{n-1}, \quad \text{where } 0 \leqslant d_i \leqslant p_i - 1 (i = 0, 1, \dots, n)$$

and d_i's are the mixed-radix digits. This algorithm essentially determines the d_i's using r_i and p_i. For convenience, let $R_k = d_0 + d_1 p_0 + \cdots + d_k p_0 \dots p_{k-1}$. We then have

$$\begin{aligned} r \bmod p_0 &= r_0 = d_0 \\ r \bmod p_1 &= r_1 = d_0 + d_1 p_0 \\ &\vdots \\ r \bmod p_k &= r_k = R_k \\ &\vdots \\ r \bmod p_n &= r_n = r. \end{aligned}$$

Thus, $d_1 = (r_1 - d_0)\ p_0^{-1} \bmod p_1$. Similarly,

$$d_2 = (r_2 - (d_0 + d_1 p_0))(p_0 p_1)^{-1} \bmod p_2$$

or

$$d_2 = (r_2 - R_1)(p_0 p_1)^{-1} \bmod p_2,$$

and in general

$$d_k = (r_k - R_{k-1})(p_0 \cdots p_{k-1})^{-1} \bmod p_k.$$

Note that $r < M = \prod_{i=0}^{n} p_i$; thus, $d_i = 0$ for $i \geqslant n + 1$.

3.1.2. Polynomial interpolation

For polynomials, we replace r by $r(x)$. We assume that $r(x)$ is an nth degree polynomial expressed uniquely by the Newton's Interpolation formula,

$$r(x) = d_0 + d_1(x - x_0) + \cdots + d_n(x - x_0) \dots (x - x_{n-1}).$$

The above proof can be directly extended to polynomial interpolation. Thus, $d_i = 0$ for $i \geqslant n + 1$.

3.2. Polynomial evaluation

The Newton's interpolation formula is very convenient for polynomial evaluation. Since $r(x) = d_0 + d_1(x - x_0) + \cdots + d_n(x - x_0) \dots (x - x_{n-1})$ we can evaluate $r(x)$ from the Newton's coefficients d_i stored in the registers by using Horner's nested multiplication rule.

4. Examples

Chinese remaindering

Let $p_i = \{5, 7, 11, 13\}$, $r_i = \{1, 5, 9, 11\}$. *Table 1* illustrates the process of reconstructing an integer.

Table 1
Chinese remaindering

	P_0 mod 5	P_1 mod 7	P_2 mod 11	P_3 mod 13	d_i	P
r_i	1	5	9	11	$d_0=1$	
$S_s := R_j$		1	1	1		
Subtract		4	8	10		
$M_s = 5^{-1}$		3	9	8	$d_1=5$	
$M_s \cdot R_s$		5	6	2		$1+5\cdot5+8\cdot5\cdot7$
$S_s := R_j$			5	5		$+7\cdot5\cdot7\cdot11$
Subtract			1	10		$=3001$
$M_s = 7^{-1}$			8	2	$d_2=8$	
$M_s \cdot R_s$			8	7		
$S_s := R_j$				8		
Subtract				12		
$M_s = 11^{-1}$				6	$d_3=7$	
$M_s \cdot R_s$				7		

4.2. Single variable interpolation - finite field

Let $x_0=1,\ r_0=8;\ x_1=2,\ r_1=3;\ x_2=3,\ r_2=0;\ x_3=4,\ r_3=10$. *Table 2* illustrates the reconstruction of the polynomial over the prime field modulo 11.

4.3. Single variable interpolation - real field

Let $x_0=1,\ r_0=0.5;\ x_1=2,\ r_1=2.5;\ x_2=3,\ r_2=6.5;\ x_3=0.5,\ r_3=0.25$. *Table 3* illustrates the reconstruction of the polynomial over the real field.

5. ISA implementation of polynomial interpolation/evaluation

We now consider the implementation of the above interpolation algorithm on the ISA, using a systematic top-down design technique. The time complexities for the ISA polynomial interpolation and evaluation programs are indicated.

Table 2
Single variable interpolation

	P_0 $x_0=1$	P_1 $x_1=2$	P_1 x_2	P_3 $x_3=4$	d_i	P
r_i	8	3	0	10	$d_0=8$	
$S_s := R_j$		8	8	8		
Subtract		6	3	2		
$(x_s-x_0)^{-1}$		1	6	4	$d_1=6$	
Multiply		6	7	8		$8+6(x-1)+1(x-1)(x-2)$
$S_s := R_j$			6	6		$=x^2+3x+4$
Subtract			1	2		(Arithmetic modulo 11)
$(x_s-x_1)^{-1}$			1	6	$d_2=1$	
Multiply			1	1		
$S_s := R_j$				1		
Subtract				0	$d_3=0$	

Table 3
Single variable interpolation

	P_0 $x_0 = 1$	P_1 $x_1 = 2$	P_2 $x_2 = 3$	P_3 $x_3 = 0.5$	d_i	p
r_i	0.5	2.5	6.5	0.25		
$S_s := R_j$		0.5	0.5	0.5	$d_0 = 0.5$	
Subtract		2	6	-0.25		
$(x_s - X_0)^{-1}$		1	0.5	-2		
Multiply		2	3	0.5	$d_1 = 2$	$0.5 + 2(x-1) + 1(x-1)(x-2)$
$S_s := R_j$			2	2		$= x^2 - x + 0.5$
Subtract			1	-1.5		
$(x_s - x_1)^{-1}$			1	-0.666		
Multiply			1	1	$d_2 = 1$	
$S_s := R$				1		
Subtract				0	$d_3 = 0$	

The systematic top-down design process for an ISA is a generalized version of the design approach for systolic arrays, see Kung [2]. We start with a locally recursive version of the above algorithm and generate the dependence graph. The final design is based on this approach, but we shall only describe the the actual implementation and not the approach (see [5]).

We have a $1 \times (n+1)$ array of $n+1$ processing elements (PE) in the ISA, $P_0, \ldots, P_n$.

Each processing element in the ISA has 6 data registers:
- X stores a point x_i
- XS shifts x_i
- R stores a residue r_i
- RS shifts r_i
- M stores M_S
- D stores D_i coefficients.

We have five instructions in the instruction set, and each PE can execute all the instructions (subscripts L and T refer to the left and top neighbours respectively). The five instructions in the ISA program are:

A. $X := X_T$, $R := R_T$
B. $R := R - \text{RS}_\text{L}$, $\text{RS} := \text{RS}_\text{L}$
C. $M := 1/(X - \text{XS}_\text{L})$, $\text{XS} := \text{XS}_\text{L}$
D. $R := R \cdot M$
E. $D := R$, $\text{RS} := R$, $\text{XS} := X$.

The ISA interpolation program for four PE P_0, P_1, P_2 and P_3 ($n = 3$) is shown in *Fig. 1*. Note that the selectors (not shown) in a one-dimensional ISA are always '1'. The input data is the set of residues r_i at points x_i ($i = 0, \ldots, n$). At the end of the program, register D in each PE P_i contains value D_i.

We observe that the pattern of instructions executed in each processing element P_i is $A(\text{BCD})^i E$, i.e. the sequence of instructions BCD is repeated i times in P_i. We can generate all possible combinations of instructions that are executed at the same time (on different PE), viz. instructions on the same horizontal line, using the following expression:

$$L = [(e, E, CB, B)(DCB)^*(D, DC, A, e)] \cup C$$

where 'e' is the empty string, 'U' is the union operation of the set of strings and '*' represents repetition of the string zero or more times.

Remark. It is not necessary to have register D for the interpolation program since the D_i coefficients are also stored in R. However, register D is necessary for the evaluation program below.

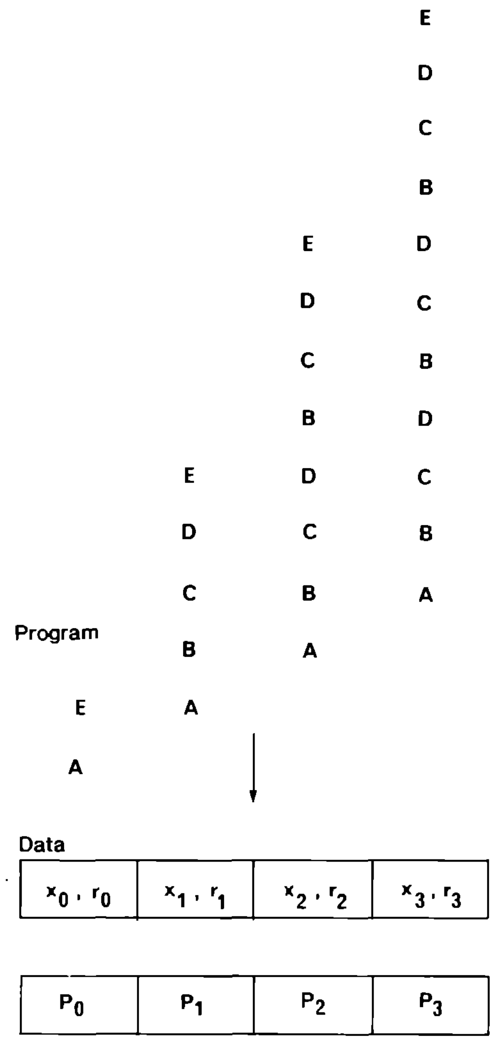

Fig. 1. ISA polynomial interpolation program.

5.1. Time complexity

Here, we have an $(m \times k)$ ISA, where $m = 1$ and $k = 4$. The period of the ISA program $= r$ = number of instructions diagonals $= 3(k-1) + 2 = 11$. The execution time of the ISA program $= r + k + m - 2 = 14$.

5.2. ISA program for polynomial evaluation

We now describe an ISA polynomial evaluation program that runs on the same $1 \times (n+1)$ processing array used for polynomial interpolation. The polynomial is evaluated at some point y as follows (X_k denotes register X in PE P_k):

$$r := D_0 + \sum_{j=1}^{n} D_j \prod_{k=0}^{j-1} y - X_k.$$

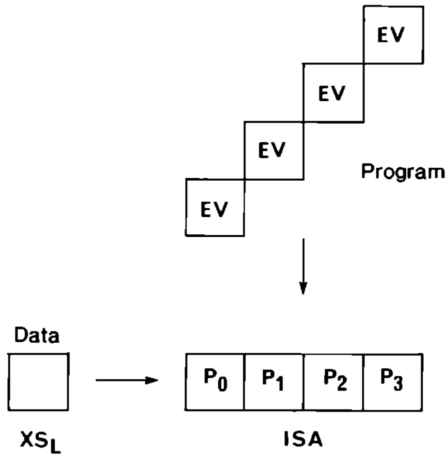

Fig. 2. ISA polynomial evaluation program.

Remark. This program can also be used to construct the nth degree polynomial interpolant r (in x) using the D_j coefficients:

$$r := D_0 + \sum_{j=1}^{n} D_j \prod_{k=0}^{j-1} x - X_k .$$

The evaluation program can be concatenated immediately after the interpolation program, as it uses the D_j coefficients (stored in the D registers) produced by the interpolation program. Also, register X in each PE P_k contains the point x_k.

The program consists of only one instruction EV. The input data is value y, the point where the polynomial is to be evaluated. This value is shifted through the processors using register XS. The program is initialized with the following values: $R_L = 0$, $M_L = 1$, $\mathrm{XS_L} = y$ (point of evaluation); these values are used when EV is executed in PE P_0.

$$EV. \quad XS := XS_L,\ M := M_L(XS_L - X),\ R := R_L + DM_L.$$

The ISA program for $n = 3$(4 PE) is shown in *Fig. 2*; the selectors are not shown.

The execution of instruction EV does not affect registers D and X. Thus, the period of this algorithm is 1, and a new point can be input at each time unit. As mentioned above, register D is necessary for the evaluation program since D_i coefficients stored in registers R are overwritten.

It is possible to reduce the number of multiplications by about one half if Horner's nested multiplication rule is used during the evaluation. However, this would change the direction of the instruction diagonal to a perpendicular direction, viz. from north-west to south-east (in contrast, the instruction diagonal for interpolation is from south-west to north-east). This will cause delay if the evaluation program is concatenated immediately after the interpolation program.

5.3. Time complexity

The period of the ISA program $= r = 1$ (number of instruction diagonals).
The execution time for an $(m \times k)$ processing array $= r + k + m - 2$. Here, $m = 1$ and $k = 4$.

Table 4
Trace of ISA program for interpolation

Time	P_0	P_1	P_2	P_3
0	A $X := x0 = 1$ $R := r0 = 0.5$			
1	E $D := R = 0.5 = d0$ RS := 0.5,XS := 1	A $X := x1 = 2$ $R := r1 = 2.5$		
2		B $R := 2.5 - .5 = 2$ RS := 0.5	A $X := x2 = 3$ $R := r2 = 6.5$	
3		C $M := 1$ XS := 1	B $R := 6.5 - .5 = 6$ RS := .5	A $X := x3 = .5$ $R := R3 = .25$
4		D $R := R.M = 2$	C $M := 1/(3-1) = .5$ XS := 1	B $R := -.25$ RS := .5
5		E D := 2 = $d1$ RS := 2,XS := 2	D $R := R.M = 3$	C $M := -2$
6			B R := 3 − 2 = 1 RS := 2	D $R := R.M = 0.5$
7			C $M := 1/(3-2) = 1$ XS := 2	B $R := .5 - 2 = -1.5$ RS := 2
8			D $R := R.M = 1$	C $M := 1/(.5-2)$ $= -2/3$
9			E $D := 1 = d2$ RS := 1,XS := 3	D $R := R.M = 1$
10				B $R := 1 - 1 = 0$ RS := 1
11				C $M := 1/(.5-3)$ $= -2/5$
12				D $R := R.M = 0$
13				E $D := 0 = d3$ RS := 0,XS := .5

Hence, execution time = 4.

Remarks. For Chinese Remaindering (CRT) the subprogram C is modified thus:

$$M := 1/XS_L, \ XS := XS_L$$

Remarks. For evaluating the result in Chinese Remaindering, the EV program is modified thus:

Initiate: $XS_L := 1;\ R_L := 0;\ M_L := 1;$

EV: $XS := X(\text{Prime}),\ M := M_L \cdot XS_L,\ R := R_L + D \cdot M_L.$

Tables 4 and *5* give the trace of ISA program for interpolation and evaluation of the polynomial given in *Table 3*. The evaluation is carried out for $x = -1$.

Table 5
Trace of ISA polynomial evaluation

Time	P_0	P_1	P_2	P_3
0	$D := d0 = .5$ $\mathrm{XS} := -1$ $X := x0 = 1$ $M := 1 \cdot -2 = -2$ $R := 0 + .5 = .5$			
1		$D := d1 = 2$ $\mathrm{XS} := -1$ $X := x1 = 2$ $M := -2 \cdot -3 = 6$ $R := .5 - 4 = -3.5$		
2			$D := d_2 = 1$ $\mathrm{XS} := -1$ $X := x2 = 3$ $M := 6 \cdot -4 = -24$ $R := -3.5 + 6 = 2.5$	
3				$d := d3 = 0$ $\mathrm{XS} := -1$ $X := x3 = .5$ $M: = -24 \cdot -1.5$ $= 36$ $R := 2.5$

Acknowledgements

The authors thank Richard Brent for his interest in this work and the Australian National University, Canberra, for providing support during the time of this work. Also the authors thank the referees for providing suggestions to improve the presentation of this paper.

References

[1] E.V. Krishnamurthy, *Error-free Polynomial Matrix Computations* (Springer, New York, 1985).
[2] S.Y. Kung, *VLSI Array Processors* (Prentice Hall, Englewood Cliffs, NJ, 1988).
[3] H.W. Lang, The instruction systolic array, a parallel architecture for VLSI, *VLSI J.* (1986) 65–74.
[4] R.J. McEliece and D.V. Sarvate, On sharing secrets and Reed–Solomon codes, *Comm. ACM* 24 (1981) 583–590.
[5] D.I. Moldavan, On the design of algorithms for VLSI systolic arrays, *Proc. IEEE* 71 (1983) 113–120.
[6] H. Schroder, The instruction systolic array – a trade-off between flexibility and speed, Technical Report CS-86-08, Department of Computer Science, Australian National University, Canberra.
[7] A. Shamir, How to share a secret, *Comm. ACM* 22 (1979) 612–614.