On Parallel Construction of Multidimensional Chains of Recurrences *

Eugene V. Zima
Comput. Math. & Cybernetics dept.,
Moscow State University,
Moscow 119899, Russia.
zima@cs.msu.su

1 Introduction

It is not rare case in parallel processing, when the quite complicated transformations are used in order to transfer the problem to most suitable for parallelization form. It is advantage of any parallelization technique if these transformations themselves are parallelizable. One of such cases is studied in this paper. Consider the following computational task.

Given a closed form functions $G_1(x_1, x_2, \ldots, x_d), \ldots, G_p(x_1, x_2, \ldots, x_d)$, starting points $x_{01}, x_{02}, \ldots, x_{0d}$ and steps $h_1, h_2, \ldots, h_d$,
compute values $G_1, \ldots, G_p$ for
$$
\begin{align*}
 x_1 &= x_{01} + n_1 h_1, \\
 x_2 &= x_{02} + n_2 h_2, \\
 &\cdots \\
 x_d &= x_{0d} + n_d h_d \\
\end{align*}
$$

(1)

Tasks like this appear in such applications as plotting mathematical curves and surfaces, animation of plots, signal processing and so on. In order to expedite the computations needed the technique based on chains of recurrences can be used. The application of this technique consists of two steps:

1. (an algebraic) conversion of the initial computational scheme into chains of recurrences,
2. (a numeric) interpreting the chain based computational scheme.

---

*Work reported herein was supported in part by the Russian Fund for Fundamental Research under Grant 96-01-01138a.*
Algorithms to construct linear and two-dimensional chains of recurrences have been considered in [1, 2, 4] together with implementation of both steps mentioned above within different Computer Algebra Systems (CAS).

It was shown in [3, 6] that function evaluation in loops using CRs is inherently parallelizable. In this paper we introduce multidimensional chains of recurrences (MCR), consider the parallel interpreting scheme of MCRs and show that MCR construction algorithm is highly parallelizable independently of the architecture used. We formulate conditions when MCR construction can be done "on the flight" within numeric application without involving CAS in the process of construction. The CR-based computations look like very careful preparation to the loop and then fast loop computations in "shift-and-operate" style. Initial assignments before the loop typically takes as much time as 20-30 steps of the loop itself. "On the flight" parallel construction of ready-to-evaluate MCRs allows to reduce these expenses essentially.

Additionally we consider particular case of the computational task (1), namely the case when \( G_j(x_1, \ldots, x_d) \) is multivariate polynomial. In this case the correspondent MCR is multidimensional analog of the table of finite differences for \( G_j \). We give parallel algorithms, which allows to get MCR for polynomial in \( c \cdot d \cdot n \) steps, where \( d \) is the dimension of given problem (1), \( n \) is maximum of degrees of \( x_l, l = 1, \ldots, d \) and constant \( c \) is reasonable small.

In Section 2 we define the MCR, describe briefly an algorithm to construct MCRs and formulate parallel MCR simplification rules. In Section 3 we define explicitly the MCR interpreting scheme. In Section 4 we give linear (in maximal degree and in dimension) algorithm to construct MCRs for multivariate polynomials.

## 2 Multidimensional chains of recurrences

Before defining MCRs we reformulate (by substituting \( x_j = x_{0j} + i_j h_j, j = 1, \ldots, d \) the initial computational task to

\[
\text{Compute } F_1(i_1, \ldots, i_d), \ldots, F_p(i_1, \ldots, i_d) \\
\text{for } i_1 = 0, 1, \ldots, n_1; \ldots; i_d = 0, 1, \ldots, n_d; \tag{2}
\]

and keep \( x_{0j}, h_j, j = 1, \ldots, d \) as parameters of the task.

Let the order \( \prec \) be defined over the set of variables \( i_1, i_2, \ldots, i_d \) such that \( i_1 \prec i_2 \prec \cdots \prec i_d \). Considering any expression which does not depend on \( i_1, \ldots, i_d \) as 0-dimensional chain of recurrences, we define a \( d \)-dimensional chain of recurrences in \( i_1, \ldots, i_d \) recursively: given functions \( \varphi_{0}(i_1, \ldots, i_{d-1}), \ldots, \varphi_{k-1}(i_1, \ldots, i_{d-1}) \), defined over \((N \cup \{0\})^{d-1}\), a function \( f_k(i_1, \ldots, i_d) \) defined over \((N \cup \{0\})^d\), and operators \( \circ_1, \ldots, \circ_k \) equal to either + or *, we call a \( d \)-dimensional chain of recurrences in \( i_1, \ldots, i_d \) the set of functions \( f_0(i_1, \ldots, i_d), f_1(i_1, \ldots, i_d), \ldots, f_{k-1}(i_1, \ldots, i_d), f_k(i_1, \ldots, i_d) \) connected in such a way, that
for $0 \leq j < k$

$$f_j(i_1, \ldots, i_d) = \begin{cases} \varphi_j(i_1, \ldots, i_{d-1}), & \text{if } i_d = 0, \\ f_j(i_1, \ldots, i_{d-1}, i_d - 1) \odot_{j+1} f_{j+1}(i_1, \ldots, i_{d-1}, i_d - 1), & \text{if } i_d > 0. \end{cases}$$ (3)

Further, to denote MCRs (3), we will use the linear notation

$$f_0(i_1, \ldots, i_d) = \Phi(i_1, \ldots, i_d) = \{i_d \odot_0, \odot_1, \odot_2, \ldots, \odot_k, f_k\}_{i_d}$$

and call an expression $\Phi$ to be $d$-dimensional CR-expression in $i_1, \ldots, i_d$ if it represents one of the following functions over $(\mathbb{N} \cup \{0\})^d$:

- a $(d-1)$-dimensional CR-expression in $i_1, \ldots, i_{d-1}$;
- $d$-dimensional CR $\{i_d \odot_0, \odot_1, \odot_2, \ldots, \odot_k, f_k\}_{i_d}$, where $f_k$ is $d$-dimensional CR-expression in $i_1, \ldots, i_d$ and $\varphi_0, \ldots, \varphi_{k-1}$ are $(d-1)$-dimensional CR-expressions in $i_1, \ldots, i_{d-1}$;
- a function $P(\Phi^{(1)}, \ldots, \Phi^{(s)})$, where $\Phi^{(1)}, \ldots, \Phi^{(s)}$ are $d$-dimensional CR-expressions.

For example, the function $i^2 \exp(2j + 1)$ has the following representation as two-dimensional CR in $j, i$:

$$\Phi(j, i) = \{i, 0, +, \{j \exp(1), *, \exp(2)\}_j, +, \{2 \exp(1), *, \exp(2)\}_j\}_i.$$ (4)

For $d$-dimensional CR

$$\Phi(i_1, \ldots, i_d) = \{i_d \odot_0, \odot_1, \ldots, \odot_k, f_k\}_{i_d}$$ (5)

we call $i_d$ to be the main variable ($\text{mvar}(\Phi) = i_d$) and keep most of notions from the theory of linear CRs [4] with putting words ”w.r.t. main variable” in proper places. In particular for the MCR (5) we call

- $k = \text{L}_{i_d}(\Phi)$ the length of $\Phi$ w.r.t. main variable,
- $\Phi$ a pure-sum MCR w.r.t. main variable, if $\odot_1 = \odot_2 = \ldots = \odot_k = +$,
- $\Phi$ a pure-product MCR w.r.t. main variable, if $\odot_1 = \odot_2 = \ldots = \odot_k = *$,
- $\Phi$ a simple MCR w.r.t. main variable, if $f_k$ is $(d-1)$-dimensional MCR (i.e. if $f_k$ does not depend on the main variable),
- $\text{comp}(j, \Phi) = \varphi_j, (0 \leq j \leq k)$ the $j$-th component of a simple CR $\Phi$, 

3
\[ \Phi_r = \{ \varphi_r, \varphi_{r+1}, \varphi_{r+1}, \ldots, \varphi_k, f_k \}_{i_d}, \ (0 \leq r \leq k) \] - \text{r-order subchain of the MCR } \Phi \text{ w.r.t. main variable} \\

and so on. Therefore, in (4) \( \text{mvar}(\Phi(j, i)) = i \); \( \Phi(j, i) \) is simple pure-sum MCR of the length 2 w.r.t. main variable; \( \text{comp}(1, \Phi(j, i)) = \{ _{i_d} \exp(1), *_{i_d} \exp(2) \} \) is one-dimensional CR in \( j \), and is simple pure-product of the length 1.

In other words \( d \)-dimensional CR is viewed as a linear CR w.r.t. main variable, whose components are \((d - 1)\)-dimensional CRs. This point of view on MCRs as on the form of internal representation of expressions is close to the recursive representation of multivariate polynomials, when for example an element of the ring \( K[i_1, \ldots, i_{d-1}, i_d] \) is considered as an element of the ring \( K[i_1, \ldots, i_{d-1}, i_d] \) (i.e. as polynomial in \( i_d \) with coefficients from \( K[i_1, \ldots, i_{d-1}] \)).

**Remark 1** This similarity between MCRs and recursive internal representation of multivariate polynomials is not occasional. If all the functions \( F_i(i_1, \ldots, i_d) \) in (2) are polynomials, MCR in \( i_1, \ldots, i_d \) gives the analog of the canonical recursive form of representation of \( F_i \). This MCR-representation has the nice distinguished feature (in comparison with usual recursive representation): it provides us with the fast scheme to compute values of the polynomials \( F_i \) \footnote{This scheme is \( d \)-dimensional analog of the table of finite differences and allows to compute values needed using only addition operations}.

Of course the changing the order of variables \( i_1, \ldots, i_d \) will change the internal representation (similarly to the polynomial case). Return to the function \( i^2 \exp(2j + 1) \). It has the following representation as two-dimensional CR in \( i, j \):

\[ \Psi(i, j) = \{ _{j} \{ 0, +, \exp(1), +, 2 \exp(1) \}, *, \exp(2) \} \], \quad (6) \]

which differs essentially from (4): \( \Psi(i, j) \) is simple pure-product MCR of the length 1 w.r.t. main variable \( j \) and \( \text{comp}(0, \Psi) \) is simple pure-sum one-dimensional CR of the length 2 w.r.t. main variable \( i \).

As algebraic operations on multivariate polynomials in recursive form are implemented using operations on univariate polynomials \([5]\), the algorithm to construct by given function \( F(i_1, \ldots, i_d) \) the MCRs can be obtained easily from the algorithm to construct linear CRs \([4]\). It starts from replacing all occurrences of \( i_i \) by simple MCR \( \{ _{i_d} 0, +, 1 \} \). Then MCR-simplifying rules are applied to the obtained MCR-expression recursively. The mentioned above MCR-simplifying rules can be easily obtained from linear CR-simplifying rules described in \([4]\). For example, let two MCRs \( \Phi, \Psi \) be given and \( u = \text{mvar}(\Phi), v = \text{mvar}(\Psi) \). Then the rule to simplify the MCR-expression \( \Phi + \Psi \) looks like:
One can observe that when $\text{mvar}(\Phi) \prec \text{mvar}(\Psi)$ in this rule and $\Psi = \{x \psi_0, +, \Psi_1\}_v$, then MCR $\Phi$ behaves as constant expression (in correspondent simplifying rule for linear CRs [4]) with respect to MCR $\Psi$. And it is again analogy of behavior of a polynomial from $K[i_1, \ldots, i_{m-1}]$ with respect to a polynomial from $K[i_1, \ldots, i_{m-1}][i_m]$ in the case of polynomials' addition.

All other simplifying rules are constructed from linear-simplifying rules in the same way. We will not rewrite all these rules for MCR, but concentrate on parallel nature of simplifying rules for pure-product and pure-sum MCRs (these rules are used most frequently in the time of MCR construction). Let $\varphi_0, \varphi_1, \ldots, \varphi_k, \psi_0, \psi_1, \ldots, \psi_k$ and $c$ be $(d-1)$-dimensional MCRs in $i_1, \ldots, i_d$. Consider two simple pure-sum MCRs in $i_1, \ldots, i_d$ of the same length $k$: 

$$\Phi = \{i_d \varphi_0, +, \varphi_1, \ldots, +, \varphi_k\}_{i_d} \quad \text{and} \quad \Psi = \{i_d \psi_0, +, \psi_1, \ldots, +, \psi_k\}_{i_d}.$$ 

Rules to obtain MCRs in $i_1, \ldots, i_d$ for $\Phi \pm \Psi$, $c \Phi$ and $\exp \Phi$ look like:

$$\Phi \pm \Psi = \{i_d \varphi_0 \pm \psi_0, +, \varphi_1 \pm \psi_1, \ldots, +, \varphi_k \pm \psi_k\}_{i_d}$$

$$c \Phi = \{i_d c \varphi_0, +, c \varphi_1, \ldots, +, c \varphi_k\}_{i_d}$$

$$\exp \Phi = \{i_d \exp \varphi_0, *, \exp \varphi_1, \ldots, *, \exp \varphi_k\}_{i_d}$$

For two simple pure-product MCRs in $i_1, \ldots, i_d$ of the same length $k$: 

$$\Phi = \{i_d \varphi_0, *, \varphi_1, \ldots, *, \varphi_k\}_{i_d} \quad \text{and} \quad \Psi = \{i_d \psi_0, *, \psi_1, \ldots, *, \psi_k\}_{i_d},$$

we have analogous rules to obtain MCRs in $i_1, \ldots, i_d$ for $\Phi * / \Psi$, $c \Phi$ and $\log \Phi$:

$$\Phi * / \Psi = \{i_d \varphi_0 * / \psi_0, *, \varphi_1 * / \psi_1, \ldots, *, \varphi_k * / \psi_k\}_{i_d}$$

$$c \Phi = \{i_d c \varphi_0, *, c \varphi_1, \ldots, *, c \varphi_k\}_{i_d}$$

$$\log \Phi = \{i_d \log \varphi_0, +, \log \varphi_1, \ldots, +, \log \varphi_k\}_{i_d}$$

It is easy to see that all componentwise operations here (like $\varphi_j \pm \psi_j$, $c \varphi_j$ or $\exp \varphi_j$) can be performed in parallel. The use of the parallel construction could be argued for linear CRs
(when each of the operations mentioned is very simple), but if \( d > 1 \) then every operation above uses recursive calls of simplifying rules for \((d - 1)\)-dimensional MCRs which is quite expensive task. That is why the possibility to perform all the componentwise operations in parallel is very attractive.

**Remark 2**  It was enough here to consider operations on MCRs of the same length w.r.t. main variable, because for any MCR
\[
\{i_d \varphi_0, \varphi_1, \ldots, \varphi_k\}_i = \{i_d \varphi_0, \varphi_1, \ldots, \varphi_k, +, 0\}_i = \{i_d \varphi_0, \varphi_1, \ldots, \varphi_k, *, 1\}_i.
\]

**Remark 3**  It does not matter which kind of operations are performed in rules above: algebraic operations on expressions, or algebraic operations on \((d - 1)\)-dimensional MCRs, because we consider the MCRs as one of the forms of the internal representation.

**Remark 4**  The difference in construction of the MCRs for the task (1) is only on the first (substitution) step, when all occurrences of \( x_t \) are replaced by \( \{i_t x_{0t}, +, h_t\}_i \).

Although, most of the MCR parallel simplifying rules are of constant complexity, there are some problems with the rule to multiply two simple pure-sum MCRs. Given two MCRs

\[
\Phi = \{i_d \varphi_0, +, \varphi_1, \ldots, +, \varphi_k\}_i \quad \text{and} \quad \Psi = \{i_d \psi_0, +, \psi_1, \ldots, +, \psi_l\}_i,
\]

the explicit formula to get components of the MCR

\[
\Delta = \Phi \Psi = \{i_d \delta_0, +, \delta_1, \ldots, +, \delta_{k+l}\}_i
\]

looks quite complicated:

\[
\delta_t = \sum_{u = \max(0, t - l)}^{\min(t, k)} \binom{t}{u} \varphi_u \sum_{v = t - u}^{\min(t, l)} \binom{u}{v} \psi_v \quad (7)
\]

for \( t = 0, 1, \ldots, k + l \). However, this formula allows us to derive useful partial case, when \( \Psi = \{i_d 0, +, 1\}_i \) (i.e. \( \Psi(i_1, \ldots, i_d) = i_d \)):

\[
\Delta = \Phi \Psi = \{i_d \varphi_0, +, \varphi_1, \ldots, +, \varphi_k\}_d \{i_d 0, +, 1\}_d = \{i_d 0, +, 1(\varphi_0 + \varphi_1), +, 2(\varphi_1 + \varphi_2), \ldots, +, k(\varphi_{k-1} + \varphi_k), +, (k + 1)(\varphi_k + 0)\}_i. \quad (8)
\]

If MCR \( \Psi = \{i_d \psi_0, +, \psi_1\}_i \) is arbitrary simple pure-sum MCR of the length 1 w.r.t. main variable \( i_d \), we can simply reduce computation of the product \( \Phi \Psi \) to the previous case using following formula:

\[
\{i_d \varphi_0, +, \varphi_1, \ldots, +, \varphi_k\}_d \{i_d \psi_0, +, \psi_1\}_d = \{i_d \varphi_0 \psi_0, +, \varphi_1 \psi_0, \ldots, +, \varphi_k \psi_0\}_d + \{i_d \varphi_0 \psi_1, +, \varphi_1 \psi_1, \ldots, +, \varphi_k \psi_1\}_d \{i_d 0, +, 1\}_d. \quad (9)
\]
The last is based on simple equality \( \{_{i_d} \psi_0, +, \psi_1 \}_{i_d} = \psi_0 + \psi_1 \{_{i_d} 0, +, 1 \}_{i_d} \), which is MCR simplifications rules just rewritten in the backward order.

Now, observing that rules to simplify products of two simple pure-sum MCRs are used to get MCR representation of polynomials, utilizing either (8) or (9) and using standard Horner scheme, we can easily get an algorithm to construct MCR for polynomial in linear in polynomial’s degree time. We will discuss this algorithm for multivariate polynomials in details in the next but one section. Here, as an example, we consider one-dimensional version of this algorithm.

Let the \( n \)-degree polynomial \( a_n i^n + \ldots + a_1 i + a_0 \) in \( i \) be given by an array \( a \) of coefficients \( (a[0], a[1], \ldots, a[n]) \). Let \( \text{mltr} \) be an array of the same length with components \( \text{mltr}[j]=j+1 \) for \( j = 0,1,\ldots,n \) and \( s, w \) be working arrays. We will use two auxiliary functions \( \text{LeftShift}(s) \) which returns an array \( s \) shifted on one component to the left and \( \text{RightShift}(s) \) which returns an array \( s \) shifted on one component to the right \(^2\). The following program constructs the array \( s \) of the components of CR for given polynomial:

\[
\begin{align*}
& s[j] := 0 \text{ for all } j=0,1,\ldots,n; \\
& s[0] := a[n]; \\
& \text{for } j := n-1 \text{ down to } 0 \text{ do} \\
& \quad w := \text{LeftShift}(s); \\
& \quad s := \text{RightShift}( (s+w)*\text{mltr} ); \\
& \quad s[0] := a[j] \\
& \text{od;}
\end{align*}
\]

Here operations \( * \) and \( + \) are componentwise and can be performed in parallel, as well as all the shifts. That is why this program has linear complexity in the degree of the given polynomial.

**Remark 5** The program above is based on the formula (8). Analogous program to convert polynomial \( a_n x^n + \ldots + a_1 x + a_0 \), where \( x = x_0 + i \ast h \) into CR will use formula (9) with \( \psi_0 = x_0 \) and \( \psi_1 = h \). It will perform a bit more instructions in the loop, but its complexity will be again linear in degree of given polynomial.

### 3 Interpreting multidimensional CRs

Given closed form function \( F(i_1, \ldots, i_m) \), after constructing the MCR \( \Phi(i_1, \ldots, i_m) \), s.t. \( \Phi(i_1, \ldots, i_m) = F(i_1, \ldots, i_m) \), we can interpret \( \Phi \) in shift-and-operate manner. In order to describe general MCR interpreting scheme we define two auxiliary functions. Let \( u \in \{i_1, \ldots, i_m\} \). We define the function \( \text{Value}_u(\Phi) \) as:

\(^2\)Remark here, that in the result of the \text{LeftShift} the first component of the argument is simply lost, and the last component of the result is equal to 0. The analogous condition holds for \text{RightShift}.

7
\[
\text{Value}_u(\Phi) = \begin{cases} 
\Phi, & \text{if } \text{mvar}(\Phi) \preceq u, \\
\varphi_0, & \text{if } \Phi = \{u \varphi_0, \odot_1, \Phi_1\}_u, \\
\{\text{Value}_u(\varphi_0), \odot_1, \text{Value}_u(\Phi_1)\}_v, & \text{if } \Phi = \{u \varphi_0, \odot_1, \Phi_1\}_v \& u \prec v, \\
P(\text{Value}_u(\Phi^{(1)}), \ldots, \text{Value}_u(\Phi^{(s)})), & \text{if } \Phi = P(\Phi^{(1)}, \ldots, \Phi^{(s)}).
\end{cases}
\]

Now we can define the result of the application of the shift operator \(E_u\) to \(\Phi\):

\[
E_u(\Phi) = \begin{cases} 
\Phi, & \text{if } \text{mvar}(\Phi) \preceq u, \\
\{u \varphi_0 \odot_1 \text{Value}_u(\Phi_1), \odot_1, E_u(\Phi_1)\}_u, & \text{if } \Phi = \{u \varphi_0, \odot_1, \Phi_1\}_u, \\
\{u \text{Value}_u(\varphi_0), \odot_1, E_u(\Phi_1)\}_v, & \text{if } \Phi = \{u \varphi_0, \odot_1, \Phi_1\}_v \& u \prec v, \\
P(E_u(\Phi^{(1)}), \ldots, E_u(\Phi^{(s)})), & \text{if } \Phi = P(\Phi^{(1)}, \ldots, \Phi^{(s)}).
\end{cases}
\]

Returning to the example (4) we can write:

\[
\Phi(j, i) = \{0, +, \{j, \exp(1), \ast, \exp(2)\}_j, +, \{j, 2 \exp(1), \ast, \exp(2)\}_j\}_i,
\]

\[
\text{Value}_u(\Phi(j, i)) = 0,
\]

\[
\text{Value}_u(\Phi(j, i)) = \{0, +, \exp(1), +, 2 \exp(1)\}_j,
\]

\[
E_j(\Phi(j, i)) = \{0, +, \{j, \exp(3), \ast, \exp(2)\}_j, +, \{j, 2 \exp(3), \ast, \exp(2)\}_j\}_i,
\]

and so on. One can see, that \text{Value}_u(\Phi) returns the current value of the MCR-expression \(\Phi\) (which is a MCR-expression of the smaller dimension in general) if \(u = \text{mvar}(\Phi)\) or replaces all MCRs with main variable \(u\) by their values in MCR-expression \(\Phi\); at the same time \(E_u(\Phi)\) shifts a MCR-expression \(\Phi\) in \(u\) direction. For parallel issue it is important that the operation \(E_u(\Phi)\) is inherently parallelizable (especially for simple sum and pure-product MCRs) and can be computed very quickly (as it was shown in linear case in \([3, 6]\)) by simultaneous performing operations \(\odot_l, l = 1, \ldots, k\) from (3).

With the help of functions \text{Value}_u and \(E_u\) we give a straightforward scheme which solves the computational task (2) when \(p = 1\) (for \(p > 1\) the scheme can be easily derived from the given below). Let \text{Initialize}(\Phi) initializes components of \(\Phi\) with actual values of \(x_{ol, l}\) and other variables involved (if needed). Then the nested loop to compute and write the values of \(\Phi(i_1, \ldots, i_m)\) looks as follows:

\begin{verbatim}
Initialize(\Phi);
for i_1 := 0 to n_1 do
    \Phi^{(1)} := \text{Value}_{i_1}(\Phi) ;
for i_2 := 0 to n_2 do
    \Phi^{(2)} := \text{Value}_{i_2}(\Phi^{(1)}) ;

for i_{m-1} := 0 to n_{m-1} do
    \Phi^{(m-1)} := \text{Value}_{i_{m-1}}(\Phi^{(m-2)}) ;
for i_m := 0 to n_m do
    write( \text{Value}_{i_m}(\Phi^{(m-1)} ) ) ;
    \Phi^{(m-1)} := E_{i_m}(\Phi^{(m-1)})
od;
\end{verbatim}

\((E_u(F(u)) = F(u+1)).\)
\begin{align*}
\Phi^{(m-2)} & := E_{im-1}(\Phi^{(m-2)}) \\
\od; \quad \cdots \cdots \\
\Phi^{(1)} & := E_{i_2}(\Phi^{(1)}) \\
\od; \\
\Phi & := E_{i_1}(\Phi) \\
\od;
\end{align*}

In particular two-dimensional case the scheme looks not so complicated:

(1) Initialize(\Phi);
(2) for \(i_1\) := 0 to \(n_1\) do
(3) \(\Psi := \text{Value}_{i_1}(\Phi)\); 
(4) for \(i_2\) := 0 to \(n_2\) do
(5) write( \text{Value}_{i_2}(\Psi) );
(6) \(\Psi := E_{i_2}(\Psi)\);
(7) \od;
(8) \(\Phi := E_{i_1}(\Phi)\);
(9) \od;

Now we give a short comment to the process of interpreting for the two-dimensional CR (4) (here \(j\) corresponds to \(i_1\) and \(i\) corresponds to \(i_2\)). After initialization we have

\[ \Phi = \{i, 0, +, \{j \exp(1), *, \exp(2)\} \}, +, \{2 \exp(1), *, \exp(2)\} \}; \]

instruction at the line (3) assigns one-dimensional CR in \(i\) to \(\Psi\), i.e.:

\[ \Psi = \{i, 0, +, \exp(1), +, 2 \exp(1)\}. \]

Then in the inner loop the values

\[ 0, \exp(1), 4 \exp(1), 9 \exp(1), \ldots \]

are computed sequentially (with the help of two floating point additions each one) by shifting the CR \(\Psi\) (line (6)). When the inner loop is finished, the MCR \(\Phi\) is shifted in \(j\) direction, (line (8)) i.e.

\[ \Phi = \{i, 0, +, \{j \exp(3), *, \exp(2)\} \}, +, \{2 \exp(3), *, \exp(2)\} \}; \]

On the next step of the outer loop instruction at the line (3) again assigns one-dimensional CR in \(i\) to \(\Psi\), i.e.:

\[ \Psi = \{i, 0, +, \exp(3), +, 2 \exp(3)\}, \]

and in the inner loop the values

\[ 0, \exp(3), 4 \exp(3), 9 \exp(3), \ldots \]
are computed, and so far.

The typical trick to solve computational task (2) on SIMD machine ([3]) is to choose \( m \leq d \) and construct MCRs in \( i_1, \ldots, i_m \). Then it is possible to write for all \( i_{m+1}, \ldots, i_d \) near every instruction like

\[
\Phi^{(2)} := \text{Value}_{i_1}(\Phi^{(1)})
\]

and

\[
\Phi^{(m-2)} := E_{i_{m-1}}(\Phi^{(m-2)})
\]

in the general interpreting scheme above. It means that every simple operation on MCR will be performed simultaneously over large amount of components. Hence in such a situation values of the components of MCRs are viewed as "hypervalue", and \( n_{m+1} \cdot n_{m+2} \cdot \ldots \cdot n_d \) MCRs are interpreted simultaneously. The possibility of such a point of view follows from flexible view on values of the MCRs components (Remark 3). This flexibility will be shown up again in the next section.

4 Obtaining MCR for polynomial in linear time

Let \( F(i_1, \ldots, i_d) \) be polynomial in \( i_1, \ldots, i_d \) represented in dense recursive form ([5]). It means, that the \( d \)-dimensional array \( A[0..n_1, \ldots, 0..n_d] \) of coefficients is given. This representation supposes that \( F \) is viewed as an element of the ring \( K[i_1] \ldots [i_{d-1}][i_d] \); an element of the ring \( K[i_1] \) is represented by linear array of coefficients from \( K \); an element of the ring \( K[i_1][i_2] \) is represented as an array of elements of \( K[i_1] \), i.e. as matrix, whose columns are representations of coefficients of \( i_2 \) as elements of the ring \( K[i_1] \) and so on.

We propose linear in \( d \) and \( \max(n_1, \ldots, n_d) \) procedure to convert this array into array which represents MCR in \( i_1, \ldots, i_d \) for \( F \). The procedure is based on the same ideas as in linear case (see the last example of the Section 2) and we will follow almost the same notation. Given \( d \)-dimensional array \( s[0..n_1, \ldots, 0..n_d] \) and \( i \in \{i_1, \ldots, i_d\} \), the function \( \text{LeftShift}_i(s) \) returns an array \( s \) shifted on one component to the left in \( i \) direction (here "left" means to the side of decreasing the index \( i \)). The function \( \text{RightShift}_i(s) \) returns an array \( s \) shifted on one component to the right in \( i \) direction (here "right" means to the side of increasing the index \( i \)). The expression \( A[i=j] \) denotes the \((d-1)\)-dimensional sub-array of the array \( A \) obtained by fixing the value of the index \( i \). Finally, the expression \((j+1)[i=j] \) denotes the \( d \)-dimensional array of the same size as \( A \), whose elements for \( i = j \) and for any values of other indexes \( i_1, \ldots, i_d \) are equal to \( j + 1 \). In other words, for fixed \( i = j \) the elements of this array are constant and equal to \( j + 1 \); at the same time the values of the elements mentioned increase (linearly, as \( j + 1 \) only) in \( i \) direction. Observing, that every operation like shifting, computing \( A[i=j] \) or \((j+1)[i=j] \) corresponds to the single parallel instruction on SIMD machine (for example CM/2), we give the converting procedure announced. It takes an array \( A \) as input and returns MCR in \( i_1, \ldots, i_d \) in the same array \( A \); arrays \( s, w, mlrt \) of the same size are used as auxiliary ones:

\[
\text{for } u \in \{1, 2, \ldots, d\} \text{ do }
\]
Consider example of running this procedure with $d = 2$. Let
\[
F(x, y) = x^3(2y^2 - y + 1) + x^2(y^3 + 2y - 1) + x(y^2 - 2) + 2y^3 + 1,
\]
is represented by matrix of coefficient
\[
A = \begin{pmatrix}
1 & -2 & -1 & 1 \\
0 & 0 & 2 & -1 \\
0 & 1 & 0 & 2 \\
2 & 0 & 1 & 0 \\
\end{pmatrix}.
\]
The first column of this matrix corresponds to the coefficient near $x^0$ (1, 0, 0, 2 are coefficients of the corresponding polynomial $2y^3+1$ in $y$), the second column corresponds to the coefficient near $x^1$ and so far. We present below consequent values of variable $s$ on the first step of the outer loop: the value of $s$ before inner loop, and values after every step of the inner loop (for $j = 2, 1, 0$).

\[
\begin{pmatrix}
2 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{pmatrix}, \quad \begin{pmatrix}
0 & 1 & 0 & 2 \\
2 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{pmatrix}, \quad \begin{pmatrix}
0 & 0 & 2 & -1 \\
2 & 1 & 1 & 2 \\
4 & 0 & 2 & 0 \\
0 & 0 & 0 & 0 \\
\end{pmatrix}, \quad \begin{pmatrix}
1 & -2 & -1 & 1 \\
2 & 1 & 3 & 1 \\
12 & 26 & 4 \\
12 & 0 & 6 & 0 \\
\end{pmatrix}.
\]

On the second step of the outer loop variable $s$ obtains consequently the following values:

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{pmatrix}, \quad \begin{pmatrix}
-1 & 1 & 0 & 0 \\
3 & 1 & 0 & 0 \\
6 & 4 & 0 & 0 \\
6 & 0 & 0 & 0 \\
\end{pmatrix}, \quad \begin{pmatrix}
-2 & 0 & 2 & 0 \\
1 & 4 & 2 & 0 \\
2 & 10 & 8 & 0 \\
0 & 6 & 0 & 0 \\
\end{pmatrix}, \quad \begin{pmatrix}
1 & -2 & 4 & 6 \\
2 & 5 & 12 & 6 \\
12 & 12 & 36 & 24 \\
12 & 6 & 12 & 0 \\
\end{pmatrix}.
\]

On the first step of outer loop all the shiftings were performed vertically (in $y$ direction) on the second step - horizontally (in $x$ direction). All the conversion takes 6 parallel additions, 6 parallel multiplications, 12 parallel shiftings (6 in each direction) and 20 parallel assignments.

**Remark 6** It is interesting to observe, that the order of processing all steps of the outer loop is unessential. It does not matter if we add components of the variable $s$ as coefficients of polynomials or as components of MCRs (see Remark 3). The only important thing here is to proceed all the dimensions of the given matrix.
The procedure above is based on recursive using of the formula (8). Analogous procedure
to convert polynomial \(G(x_1, \ldots, x_d)\) in \(x_1, \ldots, x_d\), where \(x_u = x_{0u} + i_u \ast h_u\) into MCR uses
formula (9). The complexity of converting such a polynomial into MCR in \(i_1, \ldots, i_d\) will be
again linear in \(d\) and maximal degree of given polynomial.

5 Conclusion

The parallel features of MCR-construction algorithms considered here allow to conclude,
first of all, that it is possible to construct MCR-based scheme for the problem (1) or (2) in
time, proportional to maximal degree of polynomials, occured in this task as subexpressions.

Another observation is that it is possible to utilize parallel architecture available on
the earlier steps of the application of MCR-based technique. In [6] problems of mapping
symbolically constructed CRs onto parallel architectures were discussed. Now it is possible
to map initial problem (1) or (2) on parallel architecture and construct MCR-based parallel
computational scheme on the flight. This construction can be done without any special
symbolic tools within numeric application if all parameters of the computational problem
(1) or (2) have numeric values to the time of construction.

We finish with typical SIMD example, presented in [3]. Consider the function

\[
f(x, y, t) = e^{g(x, y, t)} = e^{\frac{e^{4x^2 + 3y^2} - \frac{3y^2}{10} + \frac{t^2}{10} - 0.15ty^2 - 0.09y^2}{t}}
\]

which has to be computed for

\[
t = 0, \tau, \ldots, n\tau; x = 0, h, \ldots, mh; y = 0, h, \ldots, mh;
\]
\[
h = 0.02, \tau = 0.01.
\]

We decide to construct CR w.r.t. variable \(t\) to expedite computations and can construct
symbolically the CR

\[
f(x, y, t) = \{e^{-0.09y^2}, e^{0.25r^3x^2 + 0.25r^3xy^2 - 0.3r^2xy + 0.0625r^2y^4 - 0.15ry^3}, *
\]
\[
, e^{3.50r^4x^2 + 1.50r^4xy^2 - 0.6r^2xy + 0.1250r^2y^4}, *
\]
\[
, e^{0.0r^4x^2 + 1.50r^4xy^2}, *, e^{6.0r^4x^2}\}.
\]

It is interesting to observe that initial assignments of these components to the array for all \(x, y\)
take 0.33sec. on CM/2, when following 200-steps loop computing the values needed by means
of this CR takes 2.08sec. It is easy to see that this initialization consists of dozens massively
additions/multiplications and 5 exponentiations. If we use algorithms considered here and
map \(g(x, y, t)\) as polynomial in \(t\) on the array for all \(x, y\), we do the same initialization (with
the help of parallel construction procedure presented above) in 4 parallel multiplications, 4
parallel additions, 8 parallel shifts and one parallel exponentiation. This reduces the time of
initialization to 0.08sec.
Once we have defined interpreting scheme for MCRs, we are able to develop either stand-alone parallel interpreter of MCRs, or MCR-based code generator for parallel machines, or standalone MCR- constructor, which can be linked to any application without additional symbolic systems. The first of mentioned tools is developed at MSU for CM/2 on the C* language, while the third is under experimental implementation.

References


