RISC-Linz
Research Institute for Symbolic Computation
Johannes Kepler University
A-4040 Linz, Austria, Europe

Parallel Computation and Indefinite Summation: A ||MAPLE|| Application for the Rational Case

Roberto PIRASTU, Kurt SIEGL

(December 16, 1994)

RISC-Linz Report Series No. 94-73

Supported by: Austrian Science Foundation (FWF) project Parallel Symbolic Computation no. S5302-PHY. Commission of the European Communities in the framework of the program "Human Capital and Mobility", contract nr. ERBCHBICT930501.

Published in: Journal of Symbolic Computation, Special Issue on *Symbolic Computation in Combinatorics* Δ_1, P. Paule, V. Strehl (eds.)

J. Symbolic Computation (1995) **11**, 0–??

Parallel Computation and Indefinite Summation: A ||MAPLE|| Application for the Rational Case

Roberto Pirastu[†] and Kurt Siegl[‡]

RISC-Linz, J. Kepler University, Linz, Austria
{Roberto.Pirastu|Kurt.Siegl}@risc.uni-linz.ac.at

(Received)

The problem of computing a *closed form* for sums of special functions arises in many parts of mathematics and computer science, especially in combinatorics and complexity analysis. Here we discuss two algorithms for indefinite summation of rational functions, due to Abramov and Paule. We describe some improvements and a parallel implementation on a workstation network in ||MAPLE|| (read: parallel Maple). Our best implementation achieves a speedup of up to eight over the fastest available sequential implementation. Finally, further applications of parallel computing in this field are outlined.

1. Introduction

The problem of indefinite summation consists of finding a *closed form* for expressions like $g(n) := \sum_{k=1}^{n} f(k)$, where the summand $f(x)$ is in our case a rational function in x, or at least of computing some useful representation for studying the asymptotic behaviour of g.

Consider the field $\mathcal{K}(x)$ of rational functions over a coefficient field $\mathcal{K}$ of characteristic zero. Denote by E the usual shift operator, defined on $\mathcal{K}(x)$ by $Ef(x) := f(x+1)$, and by Δ the difference operator $\Delta := E - I$, where I is the identity operator.

The problem of indefinite summation of rational functions (IRS) is stated as follows.
Problem: *Given a proper rational function $f \in \mathcal{K}(x)$, find $h, r \in \mathcal{K}(x)$ such that*

$$f = \Delta h + r \tag{1.1}$$

and the denominator of r has minimal degree among all such decompositions.

If the pair (h, r) is a solution of the indefinite summation problem for f, then we call r a *bound* for f.

For studying sum expressions like $g(n) := \sum_{k=1}^{n} f(k)$, notice that any solution (h, r)

[†] Supported by the Commission of the European Communities in the framework of the program "Human Capital and Mobility", contract Nr. ERBCHBICT930501
[‡] Supported by the Austrian Science Foundation (FWF) Proj. Parallel Symbolic Computation, NoS5302-PHY

0747–7171/90/000000 + 00 $03.00/0

of the IRS problem corresponds to a decomposition in the following form

$$g(n) := \sum_{k=1}^{n} f(k) = h(n+1) - h(1) + \sum_{k=1}^{n} r(k) \tag{1.2}$$

In particular, a solution of (1.1) with remainder $r = 0$ provides a rational *closed form* for g as an expression in n, namely $g(n) = h(n+1) - h(1)$. Otherwise, the nonsummable part r cannot be further decomposed as in (1.1) with a remainder of smaller degree in the denominator.

As an example, consider an easy problem which often arises in basic analysis courses. We want to compute a value for

$$s = \sum_{k=1}^{\infty} \frac{1}{k(k+3)} = \frac{1}{1 \cdot 4} + \frac{1}{2 \cdot 5} + \frac{1}{3 \cdot 6} + \cdots \tag{1.3}$$

We first compute, if possible, a rational closed form for $g(n) = \sum_{k=1}^{n} \frac{1}{k(k+3)}$. It is easily checked that

$$\frac{1}{x(x+3)} = \Delta \left(-\frac{3\,x^2 + 6\,x + 2}{3x\,(x+1)\,(x+2)} \right)$$

so, we have

$$g(n) = \frac{n\left(11\,n^2 + 48\,n + 49\right)}{18\,(n+1)\,(n+2)\,(n+3)}$$

From this, it immediately follows that $s = \lim_{n\to\infty} g(n) = 11/18$.

Several algorithms are known for computing solutions of the indefinite rational summation problem (see Moenck (1977), Abramov (1975), Paule (1993), Pirastu and Strehl (1995)). Here we present parallel implementations of two algorithms due to Abramov (1975) and to Paule (1993), respectively. For the sequential implementations we refer to Pirastu (1992) and (1995).

Abramov proposed already in 1975 an approach for solving the IRS problem. We slightly modify this method to obtain some further properties of the solutions.

This algorithm permits us to decompose the problem in several subproblems, which can be solved in parallel, as we later describe.

Paule (1993) provides a general algebraic framework for indefinite summation of rational functions. Besides other results, he suggests a different algorithm, which reduces the problem to the solution of a system of linear equations, in analogy to the Horowitz method for the integration of rational functions (see, for instance, Geddes et al. (1992)). In this case the parallelization consists of the solution of the linear system using a parallel Gauß Jordan algorithm.

Strehl and the first author gave in 1995 a detailed theoretical description of the problem and a new algorithm. In particular, the question of finding *optimal* solutions with minimal degree in the denominator of the summable part is treated.

The parallel implementations are realized in ||MAPLE||, a system for parallel symbolic computation (see Siegl (1993)) based on the parallel language Strand and the symbolic computation system Maple. ||MAPLE|| permits us to write parallel programs similar to the usual Maple code and does not depend on the parallel architecture used.

In this article we want to show that ||MAPLE|| is an efficient platform for parallel symbolic computation, which can be used without deep knowledge of parallel architectures.

The IRS problem treated here as a case study, shows that significant speedups can be achieved by applying different parallel methods.

In Section 2 we give some known results about the solutions to the problem and some notations. The idea of the sequential algorithm of Paule is described informally in Section 3, while the same is done for Abramov's algorithm in Section 4. The main aspects of the parallelization of both algorithms, as well as their performance and comparisons, are reported in Section 5. In Section 6 we summarize the article, giving some hint for future work in this direction.

2. Shift Structure

Before we describe the algorithms and the parallel implementation, we give a short and informal introduction to basic facts and results. We refer to Abramov (1975), Paule (1993), Pirastu (1995), Pirastu and Strehl (1995) for complete proofs.

A fundamental role in this context is played by the *shift structure* of a polynomial. We say that two polynomials p and q in $\mathcal{K}[x]$ are *shift equivalent* if $p = E^k q$ for some integer k, and we write $p \sim q$. Consider, for a polynomial $p \in \mathcal{K}[x]$, the complete factorization (over $\mathcal{K}$) $p = \alpha p_1^{e_1} p_2^{e_2} \cdots p_m^{e_m}$, where all p_i are distinct, irreducible and monic, $e_1 \cdots e_m \neq 0$, and $\alpha \in \mathcal{K}$. Then an equivalence class of the set of irreducible factors $\{p_1, \ldots, p_m\}$ of p under the relation $\sim$ is called a *shift class* of p. A factor $\tilde{p} = p_{i_1}^{e_{i_1}} \cdots p_{i_l}^{e_{i_l}}$ of p is called a *shift component* of p if the set $\{p_{i_1}, \ldots, p_{i_l}\}$ is a shift class of p.

Example 2.1. *Consider the polynomial $p \in \mathbb{Q}[x]$ given by the following factorization*

$$p = (x^2 + 3)(x^2 + 2x + 4)x(x + 1)^2(x + 3)(x + 5)^2$$

then the set $\{x, x + 1, x + 3, x + 5\}$ is a shift class of p and $\tilde{p} = x(x + 1)^2(x + 3)(x + 5)^2$ is the corresponding shift component.

We can represent graphically the shift structure of a shift component. For instance Figure 1 represents the situation for $\tilde{p}$ of Example 2.1.

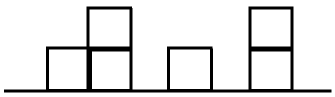

Figure 1. Shift structure of $x(x + 1)^2(x + 3)(x + 5)^2$

We fix a factor q_0, and draw d squares at the i-th position on a line when the polynomial $E^i q_0$ arises with multiplicity d as a factor of $\tilde{p}$.

Definition 2.1. *The dispersion of a polynomial p is the maximal integer distance between roots of p and is denoted by* $\mathrm{dis}(p)$*. As a convention,* $\mathrm{dis}(p) = 0$ *for constant p.*

For a proper rational function given by a *reduced representation* $f = p/q$, i. e., p and q are relatively prime polynomials, we define $\mathrm{dis}(f) = \mathrm{dis}(q)$. In Example 2.1 we have $\mathrm{dis}(p) = 5$, viz. the maximal distance between stacks.

As remarked in Abramov (1975), we have the following lemma (for a proof, see Paule (1993) or Pirastu (1992)).

Lemma 2.1. *For $f \in \mathcal{K}(x)$ let $h, r \in \mathcal{K}(x)$ be such that $f = \Delta h + r$. Then r is a bound for f if and only if* $\mathrm{dis}(r) = 0$.

From Lemma 2.1 it follows that the problem of indefinite summation is solved for decompositions $f = \Delta h + r$ where $\mathrm{dis}(r) = 0$, i.e., where each shift component of the denominator of r has only one stack of boxes in the corresponding diagram. As we will see later, both algorithms make use of this property.

Since both algorithms need the value of the dispersion of the summand f, this computation is relevant. Assume that effective algorithms are known for arithmetic in $\mathcal{K}$ and for finding integer roots of polynomials over $\mathcal{K}$. Then the dispersion of p can be computed as the maximal value of the integer roots of the resultant $\mathrm{Res}_x(E^k p, p)$, considered as a polynomial in k. This procedure is very time consuming for p of large degree.

Efficient algorithms for polynomial factorization are available for working over particular fields, like the field $\mathbb{Q}$ of rational numbers. In this case the dispersion can be easily computed from the complete factorization of p over $\mathcal{K}$ by testing all factors for shift-equivalence (see, for instance, Pirastu (1992)).

Notice that solutions (h, r) of the IRS problem are not uniquely determined by f. For instance, the decompositions

$$\Delta\left(-\frac{1}{4}\frac{2x+1}{x(x+1)}\right) - \frac{1}{2(x+2)^2} \qquad \text{and} \qquad \Delta\left(-\frac{1}{4}\frac{2x^3+7x^2+5x+2}{x^2(x+1)^2}\right) - \frac{1}{2x^2}$$

are different solutions for the same rational summand $f = \frac{1}{x(x+2)^2}$. As one can see, the degree of the denominator of h can vary considerably among solutions. In the next theorem, due to Paule, uniqueness *up to integer shifts* of the denominator of the remainders is established.

Theorem 2.1. *Let $r, r' \in \mathcal{K}(x)$ be bounds for $f \in \mathcal{K}(x)$, given by the reduced representations $r = p/q, r' = p'/q'$. If $q = q_0^{e_0} \cdots q_m^{e_m}$ is the complete factorization of q over $\mathcal{K}$, then $q' = (E^{l_0} q_0^{e_0}) \cdots (E^{l_m} q_m^{e_m})$ for some integers $l_0, \ldots, l_m$.*

In fact the main difference between the presented algorithms is the strategy for the choice of the remainder r.

If the denominator q of f splits into several shift components, viz. $q = \tilde{q}_1 \cdots \tilde{q}_s$, consider the partial fraction decomposition

$$f = \frac{p}{q} = \frac{p_1}{\tilde{q}_1} + \cdots + \frac{p_s}{\tilde{q}_s} \tag{2.1}$$

It can be shown that if (h_i, r_i) are solutions of the IRS problem for the summands $p_i/\tilde{q}_i$, then a solution for f is given by $h = h_1 + \cdots + h_s$ and $r = r_1 + \cdots + r_s$ (see, e.g., Paule (1993), Pirastu and Strehl (1995)). This means that, in our exposition, we can concentrate on the single shift components.

3. Paule's Algorithm

Paule (1993) presents an algorithm in analogy to Horowitz's algorithm for integration of rational functions (see, for instance, Geddes et al. (1992)). He reduces the problem to the solution of a system of linear equations. The idea consists of giving an "Ansatz", i. e., a candidate for the denominators of the rational solutions h and r of equation (1.1).

Let us explain the algorithms by an example: Consider the following rational function with shift structure as in the left part of Figure 2.

$$f = \frac{x^2+1}{x(x+1)^4(x+3)}$$

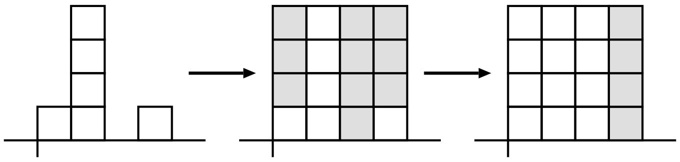

Figure 2. Paule's Ansatz for $x(x+1)^4(x+3)$

We need what Paule calls the *shift saturated extension* (SSE) of the denominator. This is an extension of the polynomial by some factors in order to let each stack of boxes in the class have the maximal arising height. In Paule (1993) and Pirastu (1992) algorithms are described, that compute such a saturation without knowing the complete factorization of the polynomials. In the example, the SSE of the denominator leads to the following representation of the rational function with shift structure like in the middle part of Figure 2:

$$f = \frac{\alpha(x)}{\beta(x)} = \frac{x^3(x+2)^4(x+3)^3(x^2+1)}{x^4(x+1)^4(x+2)^4(x+3)^4}$$

We are looking for rational solutions $h = \gamma/\delta$, $r = \varepsilon/\eta$ to (1.1). Paule proved that a solution exists for $\delta = \gcd(\beta, E^{-1}\beta)$ and $\eta = \beta/\delta$, i.e. one takes as denominator for the bound only the factors corresponding to the rightmost boxes in each component of the SSE and puts in the denominator of the summable part all other boxes. In the right part of Figure 2 one can see the decomposition in *summable* and *nonsummable* part of the SSE of the denominator.

Substituting the values for δ and η in (1.1) we obtain a polynomial equation. In our example, $\delta = x^4(x+1)^4(x+2)^4$ and $\eta = (x+3)^4$, so (1.1) becomes

$$x^3(x+2)^4(x+3)^3(x^2+1) = x^4E\gamma - (x+3)^4\gamma + x^4(x+1)^4(x+2)^4\varepsilon \tag{3.1}$$

Since we know δ and η, we have bounds for the degree of γ and ε. Substituting these polynomials in indeterminate coefficients into (3.1) and equating coefficients of same powers of x we obtain a system of linear equations for the coefficients of γ and ε.

The computation can be done by Maple. The solution of (1.1) is then $h(x) = \gamma(x)/\delta(x)$ and $r(x) = \varepsilon(x)/\eta(x)$, i.e.,

$$h(x) = -\frac{128+336x+912x^2+1764x^3+2061x^4+1491x^5+661x^6+165x^7+18x^8}{24x(x+1)^4(x+2)^4}$$

and

$$r(x) = -\frac{25 + 16x + 3x^2}{4(x+3)^4}$$

It should be remarked at this point that in general the computed solutions $\gamma(x)/\delta(x)$ and $\varepsilon(x)/\eta(x)$ are not in reduced form. In other words, the Ansatz for the denominators is not always minimal, as some factors may cancel. Although the denominator polynomial $\eta(x)$ computed by Paule's algorithm is, is some sense, optimal, the degree of the polynomial $\delta(x)$ is in general too big. In this context *optimal* means that the candidate has minimal degree among all suitable candidates, considering only the denominator of the input function. A description of an optimal Ansatz for $\delta(x)$ is given in Pirastu and Strehl (1995), together with a new algorithm for computing it.

From a computational point of view it turns out that the solution of the system of linear equations is the most time consuming task in the algorithm, taking over 80% of the total time. For this reason, in the implementation of the algorithm, we mainly concentrate on a parallel method for solving the system.

Notice that it is not necessary to solve a linear system for each shift component. Let $f = p/q$ and q splits into several shift components, say $q = \tilde{q}_1 \cdots \tilde{q}_s$. Then we can use $\delta = \delta_1 \cdots \delta_s$ as Ansatz for the denominator of h, where δ_i is the candidate corresponding to the ith component, and analogously $\eta = \eta_1 \cdots \eta_s$. Substituting in (3.1) we have to solve only one linear system for the given rational function f.

4. Abramov's Algorithm

Abramov proposes a different method for iteratively decomposing the rational function reducing the dispersion of the remainder at each step.

Let us consider again the example in the last section. We do not need any saturation of the shift structure of the denominator, but we now isolate the rightmost boxes from the rest, viz. $v = (x+3)$ and $w = x(x+1)^4$, and decompose the rational function:

$$f = \frac{x^2+1}{x(x+1)^4(x+3)} = \frac{5x^4 + 5x^3 + 15x^2 - x + 8}{24x(x+1)^4} + \frac{-5}{24(x+3)}$$

Choosing $Eu = -5/(24(x+3))$ we get a decomposition $f = \Delta u + r$ as in (1.1) with

$$f = \Delta\left(-\frac{5}{24(x+2)}\right) + \frac{-1}{24}\frac{5x^4 + 5x^3 - 9x^2 - x - 16}{x(x+1)^4(x+2)} \tag{4.1}$$

where r has dispersion two, i.e. less than the dispersion of f.

We then iterate the procedure on the remainder r, reducing the dispersion at each step. In the end we obtain either a trivial $r = 0$, and the sum has a rational closed form, or a remainder with dispersion zero, i.e., a bound for f. We then only need to sum up the partial results u, in order to obtain h such that $f = \Delta h + r$. This way we obtain the following decomposition

$$\begin{aligned} f &= \Delta\left(-\frac{5}{24(x+2)}\right) + \frac{-1}{24}\frac{5x^4 + 5x^3 - 9x^2 - x - 16}{x(x+1)^4(x+2)} \\ &= \Delta\left(-\frac{5}{24(x+2)} - \frac{5}{24(x+1)}\right) - \frac{5x^3 + 3x - 4}{12x(x+1)^4} \\ &= \Delta\left(-\frac{5}{24(x+2)} - \frac{5}{24(x+1)} - \frac{4x^3 + 9x^2 - 6x + 12}{12x^4}\right) - \frac{3x^2 - 2x + 4}{4x^4} \end{aligned}$$

If we look more closely at the shift structure of r after each step, as in Figure 3, then we see that we shifted the rightmost stack of boxes one place to the left at each iteration, while the erased stack is included in the rational part.

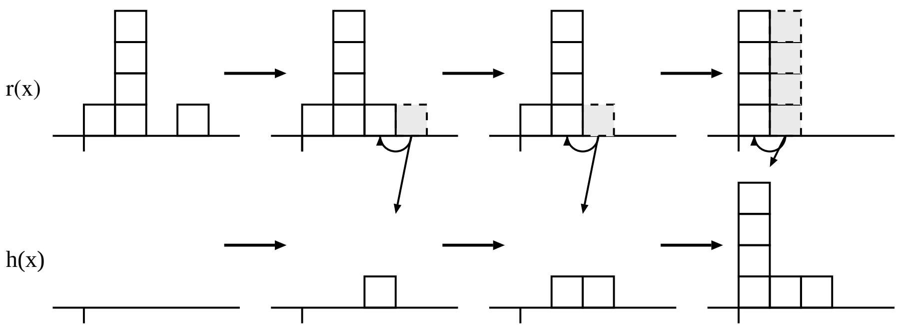

Figure 3. Shift-structure of the sub-results in Abramov's algorithm

Observe that we obtain a bound with the leftmost boxes in the shift structure of the denominator, while Paule produces a bound with the rightmost ones. On the other hand, we would obtain exactly the same result by a slight modification of Paule's method, taking the leftmost boxes as Ansatz for the bound, i.e., $\delta = \gcd(\beta, E\beta)$.

The modification we propose is based on the simple observation that in a similar way the dispersion can be reduced erasing the leftmost, instead of the rightmost, boxes of each component. This way we do not need to fix the stack corresponding to the remainder as being the leftmost from the beginning on, but we can choose at each step at which endpoints we want to erase/shift a stack of boxes. As a strategy we propose to shift at each step the stack (right or left) of smallest height, this means the stack which produces the smallest contribution to the degree of the denominator of the rational part. The following short Maple procedure `abrsum` does the job.

The procedure call `part(p,h,v,w)` computes the h-part of p, i.e., the factor v of p such that $\gcd(p/v, h) = 1$ and only factors of h arise in v. This approach has two main advantages in practice: The decomposition of the rational function is in general easier if the degree of one part is lower and the degree in the denominator of the rational part h at the end is in general smaller than by fixed choice of the remainder. In Pirastu (1994) we show that, for a certain class of rational summands, the denominator of the rational part obtained in this way has minimal degree among all solutions.

Already in our small example we obtain with the usual Abramov algorithm a rational part with denominator of degree 6, while the extension keeps the degree at 3, as we see in Figure 5. Note that Paule's algorithm computes a solution with a rational part of degree 9.

In our example the denominator q of f consists of only one shift component. In the case where q splits into more than one shift component the procedure can be applied in the same way, giving as dispersion the maximal dispersion among all shift components.

The parallelization is done by considering each shift component separately. In the implementation part of this article we describe how this can be done in ||MAPLE||.

```
part := proc (p,h,v,w)  local g,v1,w1;
   g:=gcd(p,h);v1:=1;w1:=p;
   while g<>1 do divide(w1,g,'w1');
                 v1:=v1*g; g:=gcd(w1,v1); od;
   v:=v1;w:=w1;
end:
abrsum := proc( f, dis, x)  local a,b,p,q,cp,vp,wp,vm,wm,u,newf;
   if f=0 then RETURN(0) fi;
   if dis=0 then RETURN ('Sum'( factor( f), x)) fi;
   p:=numer(f); q:=denom(f);
   cp:=gcd(q,subs(x=x+dis,q));
   if cp=1 then RETURN(abrsum(f,dis-1,x)) fi;
   part(q,cp,'vp','wp'); part(q,subs(x=x-dis,cp),'vm','wm');
   if degree(vm,x)>degree(vp,x) then
        gcdex(vp,wp,p,x,'b','a');
        u:=subs(x=x-1,a/vp); newf:=normal(b/wp+u);
   else gcdex(vm,wm,p,x,'b','a'); u:=-a/vm;
        newf:=normal(b/wm + subs(x=x+1,a/vm));
   fi;
   u + abrsum(newf, dis-1, x);
end:
```

Figure 4. Maple implementation of the Abramov's algorithm (extended)

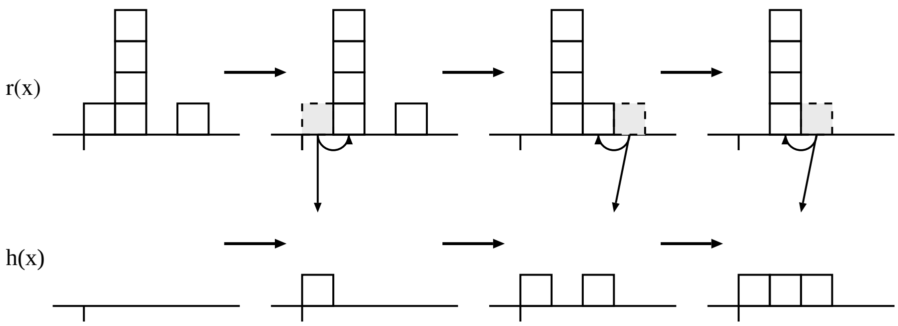

Figure 5. Shift-structures in the extended Abramov's algorithm

5. Parallel Implementation

The algorithms have been implemented in ||MAPLE|| (see Siegl (1993)) which is a portable system for parallel symbolic computation. The core of the system is built on top of an interface between the parallel declarative programming language Strand (see Foster and Taylor (1989)) and the sequential computer algebra system Maple (see Char et al. (1983)), thus providing the elegance of Strand and the power of the existing sequential algorithms in Maple.

||MAPLE|| programs may run on different hardware, ranging from shared-memory machines over distributed memory architectures up to networks of workstations, without any modification or recompilation. All necessary communication is done automatically by the system without any additional programming effort. Since ||MAPLE|| uses implicit

parallelism, it allows writing parallel programs without any expert knowledge in parallel programming.

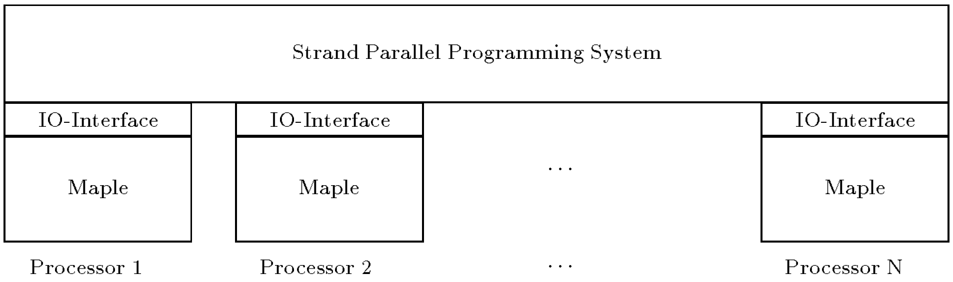

Figure 6. Structure of the ||MAPLE|| system

The ||MAPLE|| system has two layers (Figure 6). The top layer is the parallel declarative programming language Strand which controls the parallel execution of an algorithm. For performing sequential tasks we may call arbitrary Maple functions or sequences of Maple statements in the underlying Maple system over some interface routines. The result is a parallel programming system with the full functionality of Maple and parallel power of Strand.

Usually, ||MAPLE|| programs reflect this structure and consist of two parts. The Strand code for the administration of the parallel tasks, and the Maple code for the functions to be executed sequentially. On the other hand, ||MAPLE|| has a set of pre-parallelized functions for some typical algorithmic structures, which allow us to write parallel programs directly as Maple code, without the use of Strand, as in the application presented here.

Here we use a function called `peval`, designed for algorithms following the divide and conquer principle. The function may be used for a wide range of algorithms in computer algebra, in particular for recursively defined algorithms.

The ||MAPLE|| call `peval( [ f1(x1), ...., fn(xn) ], recompose)` requires two arguments:

1 A list of unevaluated functions `f1(x1)...fn(xn)`. Each of these functions will be evaluated on different processes/nodes in parallel. Note that the functions `fi` might have more than one input parameter.

2 A composition function `recompose` which takes as input the results produced by the parallel evaluations and produces the final output of the `peval` call. The subresults are contained in `parg[1],...,parg[n]`.

In particular, the functions `fi(xi)` do not need to represent the same computations and may contain other `peval` statements, and so on.

As an example for the usage of the `peval` function we give a parallel version of the well known Karatsuba algorithm for multiplying long integers in Figure 7.

Here we execute the recursive sub-multiplications in parallel as long as both integers have more than 50 digits, otherwise we call the built-in sequential version.

```
imult:=proc(x,y) local lx,ly,n,x1,x2,y1,y2;
  lx:=length(x); ly:=length(y);
  if (lx > 50) and (ly > 50)
   then                # Parallel Karatsuba algorithm for large numbers
     n:=round(max(lx,ly)/2);
     x1:=iquo(x,10^n,'x2');
     y1:=iquo(y,10^n,'y2');

     peval(['imult'(x1,y1),'imult'(x2,y2),'imult'(x1+x2,y1+y2)],
           '<u*10^(2*n)+(w-u-v)*10^n+v | n,u,v,w>'
           (n,parg[1],parg[2],parg[3])
          );
   else                # Built in algorithm for small numbers
     x*y;
  fi;
end;
```

Figure 7. Parallel integer multiplication

5.1. Paule's Algorithm

As we already saw, the most important task in the implementation of Paule's algorithm is the solution of a system of linear equations. For this reason the main goal is to improve this part of the computation.

Solving a system of linear equations in general is a well-known task. In the sequential case, the Gaussian elimination algorithm (see, for instance, Aho et al. (1974)) will be used. But it turns out that the equation solver known as Gauß Jordan algorithm allows a better parallelization. Here we successively eliminate all elements of a column in parallel, which should give us a nearly optimal speedup for larger matrices up to a high number of processors.

While with fixed-size coefficients the time required per element is constant, the elimination time with symbolic entries will vary and generate unbalanced execution times for individual rows, thus limiting the benefits of parallelism. Due to worse complexity, the Gauß Jordan algorithm is usually slower than the Gaussian algorithm by a factor of 3, so we need at least a speedup by 3 to compensate the algorithmic disadvantages. Additionally, an efficient execution with symbolic entries depends heavily on a few small details which may improve the execution time up to factor of 50 and more:

1 The smallest element in the row with minimal total size should be selected as the pivot element. In our implementation we used the amount of memory required for an element to determine its size.
2 Coefficients should always be simplified and normalized to integers.
3 Representing rows of the matrix as polynomials over a set of new variables will give us access to the highly optimized polynomial operations available in Maple.
4 In a parallel environment, communication may be optimized by grouping several rows together to form a computation block.

The effect of all these optimizations is shown by the following table comparing several built-in Maple algorithms with our implementation on a relatively small matrix with dimension 78.

Algorithm	Time	Data Type	Pivot Search
Gauß Jordan	139 min	Matrix	none
Gauß	24 min	Matrix	row pivoting
Solve (Gauß)	4 min	Polynomials	min element in min row
Our Gauß Jordan	13 min	Polynomials	min element in min row
Parallel	2 min	Polynomials	min element in min row

The table shows that by parallelism and a few other improvements the Gauß Jordan method is able to beat a highly optimized Gauß elimination algorithm by a significant factor using a couple of workstations.

We tested our algorithm on several rational functions with rational coefficients. In Figure 9 we summarize the most important timings using a network version of ||MAPLE|| on a cluster of 12 Silicon Graphics (SGI) workstations. All computation times are given in the form min:sec.

The first column shows the shift structure of the denominator, viz. $2, 5, 9$ means that the denominator of the input has three shift classes, respectively of dispersion 2, 5 and 9, respectively. The dimension of the system is given in the ninth column, while the columns six and seven are the times needed by our parallel implementation running on 12 processors or on a single processor. It should be remarked that among the 12 processors only 11 are involved in the real computations, as the first is used as manager.

The implementation using the built-in solver in Maple is reported in the eighth column, while in the last we give the obtained speedup. The second column of the speedup entries represent the speedup between the parallel implementation of Paule's algorithm on 12 processors and the sequential reference implementation.

5.2. Abramov's Algorithm

The parallelization of Abramov's algorithm lies mainly in the decomposition of the rational function with respect to the shift classes of the denominator as in (2.1).

After this we apply the procedure on each of the summands $p_i/\tilde{q}_i$ from (2.1) in parallel. In the end we only need to sum up the results obtained by each parallel function call.

First we compute the shift structure the denominator q of f, say $q = \tilde{q}_1 \cdots \tilde{q}_s$. Remark that this computation is also done by our implementation of Paule's algorithm.

Then we apply the divide-and-conquer principle in parallel using the ||MAPLE|| function **`peval`**, as in the following ||MAPLE||-code.

The function **`shift_structure`** computes the shift structure of $f = p/q$, i.e. the decomposition into shift components of the denominator $q = \tilde{q}_1 \cdots \tilde{q}_s$. In **`decompose`** the number of shift components is checked. If q has only one shift component, then the sequential procedure **`abrsum`** is applied, computing a solution of the IRS problem for f.

Otherwise, if $s \geq 2$, then the function **`shift_par_frac`** determines a decomposition

$$f = f_1 + f_2 = \frac{p_1}{\tilde{q}_1 \cdots \tilde{q}_t} + \frac{p_2}{\tilde{q}_{t+1} \cdots \tilde{q}_s}$$

where $t = \lfloor \frac{s}{2} \rfloor$. This is done by the extended Euclidean algorithm, computing p_1, p_2 such that $p_1 \cdot \tilde{q}_{t+1} \cdots \tilde{q}_s + p_2 \cdot \tilde{q}_1 \cdots \tilde{q}_t = p$. Then **`decompose`** is applied in parallel to f_1 and f_2 by

```
abrpar:= proc(f,x) local fs;
   fs:= shift_structure(f,x);
   decompose(fs,x);
end:

decompose := proc(f,x) local f1, f2;
   if nb_shift_comp(f,x)=1 then RETURN(abrsum(f,dis(f,x),x)) fi;
   shift_par_frac(f,x,'f1','f2');
   peval(['decompose'(f1,x),'decompose'(f2,x)],
         'sum_up'(parg[1],parg[2]));
end:
```

Figure 8. Parallel code for Abramov's algorithm

a `peval` call. The function `sum_up` just combines the results, summing up the summable and the nonsummable parts of the partial solutions.

In this kind of parallelization the number of really concurrent processes is directly related to the number s of shift components of q, in contrast to the solution of a linear system.

The timings for the same examples used for Paule's algorithm are given in columns two to five of Figure 9.

The entries are analogous to those corresponding to the algorithm of Paule. From the third and fourth columns follows the unexpected but interesting fact that the parallel implementation carried out on one processor is already faster than the sequential algorithm. Note that the sequential implementation considers all classes at once, so it does not need to decompose the rational function with respect to the shift classes of the denominator.

As a result of parallel considerations, this consideration implies that also a sequential implementation should compute the partial fraction decomposition and apply the procedure on the single components (at least over the coefficient field $\mathbb{Q}$ considered in our examples).

5.3. Comparison

Summerizing the data in Figure 9, for our examples the implementation of Abramov's algorithm is faster, in both the sequential and parallel case.

In the table one also finds the value of the degree in the denominator of the rational part computed by each algorithm. Our modification of Abramov's algorithm often computes a result with significantly smaller degree.

We remark that the system solver used in Paule's algorithm would take considerable advantage of having more processors available. Since the number of parallel processors used by Abramov's algorithm is given by the input, the implementation does not take any advantage of more processors available. This means that by an appropriate number of processors, Paule's algorithm would be faster in some cases, e.g., for rational functions with few shift classes, almost shift saturated structure, and corresponding to a system of high dimension.

Shift classes	Abramov				Paule					Speedup
	12 P	1 P	Seq	Deg	12 P	1 P	Seq	Dim	Deg	Abr. / Paule
2,5,9	12	19	1:18	42	2:31	13:31	4:27	78	58	6.4 / 1.7
	53	1:16	3:38	53	7:58	56:42	16:06	92	67	4.0 / 2.0
	25	45	2:49	50	18:10	143:54	38:59	111	50	6.7 / 2.1
4,10,13	1:06	1:21	3:29	81	7:46	42:22	17:14	110	84	3.1 / 2.2
	30	45	3:31	69	15:12	106:50	43:21	123	100	6.9 / 2.8
2,2,7,12	15	21	59	28	1:59	10:36	2:48	65	44	3.9 / 1.4
	9	13	28	33	1:38	5:31	3:11	69	44	3.0 / 1.9
1,5,10,11	9	18	48	44	9:14	34:10	11:48	89	44	4.8 / 1.3
	5	9	16	52	1:09	3:21	1:50	66	53	2.8 / 1.5
3,3,7,12	8	21	1:17	33	10:03	73:16	26:11	99	57	8.9 / 2.6
2,2,2,3,3,3	1:30	2:01	3:41	31	3:37	25:33	16:50	66	31	2.4 / 4.6
	1:16	1:59	6:33	46	17:55	164:44	17:12	84	46	5.1 / 0.9

Figure 9. Timings

6. Conclusions

Indefinite summation of rational functions represents a significant special case of the problem of finding closed forms for sums. We saw that within this framework it is possible to decompose the sum obtaining a sort of "simplification" even when the expression does not allow a closed form.

The algorithms discussed here showed some interesting combinatorial aspects by themselves, which can be intuitively represented by the described graphical representation.

Both algorithms suggest the use of parallel computation. The parallel implementation in ||MAPLE|| on a workstation network achieved significant speedups. Furthermore, in the case of Abramov's method the parallel approach should be used also for a sequential implementation.

This encourages us to apply parallel methods to other symbolic computation problems in combinatorics. For instance, consider the case of indefinite summation of hypergeometric functions f, i.e., where Ef/f is a rational function in x. This problem can be solved by Gosper's algorithm (see Gosper (1978) and Lisoněk et al. (1993)), which is based on the solution of a linear system with polynomial entries. Also, Zeilberger's algorithm for definite hypergeometric summation (see Zeilberger (1991)) would take advantage of an efficient parallel solver for systems of linear equations with symbolic entries.

||MAPLE|| is an efficient tool for dealing with such problems, providing the symbolic manipulation facilities of Maple. In particular, the parallel solver of linear systems presented here works without any modification for any field $\mathcal{K}$ which can be manipulated in Maple, for instance for $\mathcal{K} = \mathbb{Q}(x_1, x_2, \ldots, x_n)$, the field of rational functions in the symbols $x_1, \ldots, x_n$.

References

Abramov, S. A. (1975). *The Rational Component of The Solution of a First-Order Linear Recurrence Relation with a Rational Right Side*, Zh. vychisl. Mat. mat. fis., 15, N. 4, 1035-1039.

Aho, A.V., Hopcroft, J.E. and Ullman, J.D. (1974). *The Design and Analysis of Computer Algorithms*, Addison-Wesley.

Char, B.W., Geddes, K.O., Gentleman, W.M. and Gonnet, G.H. (1983). *The Design of Maple: A Compact, Portable and Powerful Computer Algebra System*, in EUROCAL '83, Proceedings of the ISSAC, pag. 101-115, Springer.

Foster, I. and Taylor, S. (1989). *Strand - New Concepts in Parallel Programming*, Prentice-Hall.

Geddes, K.O., Czapor, S.R. and Labahn, G. (1992). *Algorithms for Computer Algebra*, Kluwer, Boston.

Gosper, R.W. (1978). *Decision Procedure for Indefinite Hypergeometric Summation*, Proc. Nat. Acad. Sci. U.S.A., 75:40-42.

Lisoněk, P., Paule, P. and Strehl, V. (1993). *Improvement of the Degree Setting in Gosper's Algorithm*, J. of Symb. Comp. (*16*), 243–258.

Moenck, R. (1977). *On computing closed forms for summations*, Proceedings of MACSYMA Users Conference, Berkley, pp. 225-236.

Paule, P. (1993). *Greatest Factorial Factorization and Symbolic Summation I*, RISC-Linz Report Series 93-02, J. Kepler University, Linz.

Pirastu, R. (1992), *Algorithmen zur Summation rationaler Funktionen*, Diploma thesis, Univ. Erlangen-Nürnberg. (in German).

Pirastu, R. (1994). *A Note on the Minimality Problem in Indefinite Summation of Rational Functions*, in J. Zeng, editeur, Actes du Séminaire Lotharingien de Combinatoire, 31e session, Saint-Nabor, Publications de l' I.R.M.A. 1994/021, pp. 95-101.

Pirastu, R. (1995). *Algorithms for indefinite summation of rational functions in Maple*, RISC-Linz Report Series 95-09, J. Kepler University, Linz. To appear in *The Maple Technical Newsletter*.

Pirastu, R. and Strehl, V. (1995). *Rational Summation and Gosper-Petkovšek Representation*, J. Symb. Comp., this volume.

Ross, B. (1987). *Methods of Summation*, Descartes Press Co. Koriyama.

Siegl, K. (1993). *Parallelizing Algorithms for Symbolic Computation Using* ||MAPLE||, in Fourth ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming, San Diego, ACM Press, 1993, pp. 179–186. Also: RISC-Linz Report Series 93-08, J. Kepler University, Linz.

Zeilberger, D. (1991). *The Method Of Creative Telescoping*, J. of Symb. Comp. (*11*), 195–204.