NUMERIC CODE OPTIMIZATION IN COMPUTER ALGEBRA SYSTEMS AND RECURRENT RELATIONS TECHNIQUE

E.V.Zima
Department of Computational Mathematics and Cybernetics
Moscow State University, Moscow, 119899, Russia
e-mail: zima@lvk.cs.msu.su

1 Introduction

An important problem of symbolic-numeric interface is the optimization of computations generated by formulae that are obtained in computer algebra system [1]. This problem concerns not only the case of numeric code generation, because necessity in numeric computations can appear immediately in computer algebra system. It often means that large-scale scalar computations in cycles must be evaluated.

Cycles that require numeric computation can appear in the program immediately as loop statements. For example in the time of symbolic-numeric integration [2]. Another case of the numeric cycle appearance is the case of 2D or 3D plotting by expressions that are obtained in the computer algebra system. The time for 3D plotting by large expressions is often unacceptable [3]. Optimization of computations in the cycles is the main problem that is considered in this paper.

Computer algebra systems (such as Reduce, Maple, etc.) have flexible tools for numeric programs generation [4, 5, 1]. Some optimizing transformations can be performed when the code has been generated. In SCOPE package for Reduce and in Maple these transformations consist in finding of common subexpressions of arithmetic expressions given and in the reduction of computational complexity on this basis. Specialized systems for code generation (such as ALPAL [6]) have more wide collection of optimizing transformations, including transformations of cycles. However, collection of cycles' transformations in ALPAL consists of loop fusion and constant folding (or code motion in the terms of [7]) only.

When numeric computations are performed in the cycle it is natural to reorganize these computations so that each iterative step might use the results of previous steps as completely as possible. It can be done if we could find by given iterative formulae the recurrent relations that bring about the same results but economize the arithmetic operations. The simplest recurrent relations

$$h \cdot i = h \cdot (i-1) + h, x^i = x^{i-1} \cdot x, i! = (i-1)! \cdot i$$

are often used by programmers. The methods were given in [8,9] that provide the recurrent relations construction by given iterative formulae and optimized cycles' generation. In this paper we will consider combination of the recurrent relations technique with SCOPE facilities and spreading of this technique on the cycles that include calls of user's functions and procedures.

2 Recurrent relations technique and SCOPE

Let the function $f(i)$ is specified by the arithmetic expression $F(i)$. Also, let m and $n, m \leq n$, be given integers, and assume that we have to compute the values of the function for $i = m, m+1, \ldots$. Direct evaluation of $f(i)$ with the next value of i is often not economic, since not all the results of previous computational steps are utilized (some results are simply lost).

Systems of recurrent relations (SRR) of the type

$$f_{t-1}(i) = \begin{cases} \varphi_{t-1}, & i = m \\ f_{t-1}(i-1) \odot_t f_t(i-1), & i > m \end{cases} \qquad (1)$$

$$t = 1, 2, \ldots, k;$$

(here $\odot_t \in \{+, *\}$, and $f_k(i) = \varphi_k$ or $f_k(i) = G(g_1(i), \ldots, g_v(i))$, where $g_j(i), j = 1, \ldots, v$ are defined by the similar systems) and the systems of recurrent relations of the type

$$s_{t-1}(i) = \begin{cases} \xi_{t-1}, & i = m \\ s_{t-1}(i-1)c_t(i-1)+ & \\ \qquad c_{t-1}(i-1)s_t(i-1), & i > m \end{cases}$$

(2)

$$c_{t-1}(i) = \begin{cases} \psi_{t-1}, & i = m \\ c_{t-1}(i-1)c_t(i-1)- & \\ \qquad s_{t-1}(i-1)s_t(i-1), & i > m \end{cases}$$

$$t = 1, 2, \ldots, p;$$

(here $s_p(i) = \xi_p, c_p(i) = \psi_p$ or $s_p(i) = F_1(g(i)), c_p(i) = F_2(g(i))$, where $g(i)$ is defined by the system of (1) type) are being considered in [8, 9].

The integer k will be called the depth of SRR (1) and denoted by $Dp(f_0(x,i))$. Note that the depth of system (1) defines the number of operations $\odot_j$, which must be performed to obtain the next value of $f_0(i)$, and the amount of memory, required to store the intermediate results. Such systems are completely determined by the set of values $\varphi_0, \ldots, \varphi_k$ and the values of operations $\odot_1, \ldots, \odot_k$. Therefore, for shorting we shall write such systems in linear form:

$$f_0(x,i) = \{m, \odot_1, \varphi_0, \odot_2, \varphi_1, \ldots, \odot_k, \varphi_{k-1}, \varphi_k\}$$

The systems of the type (1) are arise when expression $F(i)$ contains the polynomials of i, exponents or factorials of such polynomials, etc. The systems of the type (2) are arise when expression $F(i)$ contains trigonometric functions of such polynomials. The methods of organization of computations in cycles using such SRRs are implemented in Reduce. When the cycle is optimized using SRR technique many expressions with common subexpressions are appear in front of cycle as a rule.

Example 1. Let's consider the cycle for computation the sum of the values

$$f(x) = e^{\frac{-x^2}{2}} \sin(x^3) 2^{x^3+x} \ (x = ih, \ i = 0, 1, \ldots)$$

Assume that we need bigfloat computations with precision 20 in Reduce. Program for direct computation of the sum of values $f(x)$ for $x = 0, h, \ldots, nh$ will look as following:

```
% Section A
on bigfloat, numval; precision 20;
 b:=1.0$ h:=b/n$ s:=0$
 for i:=0:n do
   <<
       x:=i*h$
       s:=s+exp(-x**2/2)*sin(x**3)*2**(x**3+x)
   >>$
```

The SCOPE package can not be applied in this case to whole program because it can not process the cycles. But we can apply SCOPE to the cycle's body. As the result of this action we will get the following cycle:

```
% Section B
 for i:=0:n do
 <<
      X:=H*I;
      G3:=X*X;
      G5:=G3*X;
      S:=S+2.0**(G5+x)*SIN(G5)*EXP(-(G3/2.0))
 >>$
```

In the result of SRR construction for cycle in section A (or B) we get the system of recurrent relations of the type (2) in which $p = 3$ and expressions ξ_j for $j = 0, 1, \cdots, 3$ are replaced by

$$\begin{aligned} &0, \\ &\sin(h^3)\exp(\frac{-h^2}{2})2^{(h^3+h)}, \\ &\sin(6h^3)\exp(-h^2)2^{6h^3}, \\ &\sin(6h^3)2^{6h^3} \end{aligned}$$

and ψ_j for $j = 0, 1, \cdots, 3$ are replaced by

$$\begin{aligned} &1, \\ &\cos(h^3)\exp(\frac{-h^2}{2})2^{(h^3+h)}, \\ &\cos(6h^3)\exp(-h^2)2^{6h^3}, \\ &\cos(6h^3)2^{6h^3} \end{aligned}$$

respectively. The program for computation of values needed now looks as following:

```
% Section C
<<
    s0:=0$ s1:=sin(h**3)*exp(-h**2/2)*2**(h**3+h)$
    s2:=sin(6*h**3)*exp(-h**2)*2**(6*h**3)$
    s3:=sin(6*h**3)*2**(6*h**3)$

    c0:=1$ c1:=cos(h**3)*exp(-h**2/2)*2**(h**3+h)$
    c2:=cos(6*h**3)*exp(-h**2)*2**(6*h**3)$
    c3:=cos(6*h**3)*2**(6*h**3)
>>$

  s:=s0$

% Section D
for i:=1:n do
  <<
     W:=S1*C0+C1*S0;
     C0:=C0*C1-S1*S0;
     S0:=W;
```

```
      W:=S2*C1+C2*S1;
      C1:=C2*C1-S2*S1;
      S1:=W;
      W:=S3*C2+C3*S2;
      C2:=C3*C2-S3*S2;
      S2:=W;
      S:=S+S0
  >>$
```

It's obviously, that assignments in front of cycle have many common subexpressions. In the result of SCOPE's operator *optimize* application to the section C we get the following section that can be substitute on the place of section C:

```
% Section E
  <<  S0:=0.0;
      G11:=H*H;
      G1:=G11*H;
      G17:=2.0**(G1+H)*EXP(-(G11/2.0));
      S1:=G17*SIN(G1);
      G9:=6.0*G1;
      G10:=EXP(-G11);
      G12:=2.0**G9;
      G16:=G12*SIN(G9);
      S2:=G16*G10;
      S3:=G16;
      C0:=1.0;
      C1:=G17*COS(G1);
      G15:=COS(G9)*G12;
      C2:=G15*G10;
      C3:=G15  >>$
```

The time for performing of sections above for n=10,20,30,40,50 is given in Appendix.

3 Optimization of the cycles that contain functions and precedures calls

Let's consider the problem of optimization of the cycles that contain calls of functions and procedures declared in the same program. Such a program can appear in the time of code generation using Maple or Reduce (Gentran) facilities. But it may be not only program on FORTRAN or Pascal that obtained in computer algebra system. We should take into account such routine of Maple as *optimize/makeproc* [10] which makes it possible to generate the Maple procedure (this procedure can be used later in another places of a Maple program, for example in cycles).

The transformation often used for program optimization is substitution of the procedure body on the place of procedure's call (including substitution of actual parameters on the place of formal parameters). When this transformation is performed for the cycle, another optimizing transformation (such as code motion, transformation of inductive variables [7]) can be applied to this cycle. However, if procedure contains large-scale expressions such transformation of the cycle will make the cycle hard to understand.

When SRR's technique is used for cycle's optimization we can avoid such substitutions. If some actual parameter of procedure is connected with system of recurrent relations we can connect this system with corresponding formal parameter and continue the process of SRR's construction in procedure's body. After that we can organize computation using the side effect and new obtained SRRs.

Example 2. Let's consider the following fragment of Pascal program:

```
 function f(x,y: real): real;
  var u, v: real;
  begin
   u:=exp(x*x/4-3*x)*y*y/2;
   v:=exp(2*x*x)/(y*y);
   f:=u/(v+1)
  end;

begin
      ...
  h:=b/n; s:=0;
  for i:=0 to n do
    begin
      x:=i*h;
      y:=exp(-x*x*x + 3*x*x -12*x);
      s:=f(x,y)+s+y
    end;
      ...
end.
```

In the result of cycles optimization, that based on SRR techique, variables x and y are connected with SRR of the type (1):

$$\begin{aligned} x &= \{0,+,0,h\} \\ y &= \{0,*,1,*,\exp(-h^3+3h^2-12h), \\ &\qquad *,\exp(-6h^3+6h^2),\exp(-6h^3)\} \end{aligned}$$

The program obtained in the result of cycle optimization is given below:

```
 function f(x,y: real): real;
  var u, v: real;
  begin
   u:=exp(x*x/4-3*x)*y*y/2;
   v:=exp(2*x*x)/(y*y);
   f:=u/(v+1)
  end;
begin
       ...
  h:=b/n; s:=0;
  { Initialization of computations using SRRs }
  y:=1; y1:=exp(-h*h*h+3*h*h-12*h);
  y2:=exp(-6*h*h*h+6*h*h); y3:=exp(-6*h*h*h);
  x:=0; x1:=h;

  s:=f(x,y)+s+y;
  { The resulting cycle }
  for i:=1 to n do
    begin
      x:=x+x1;
      y:=y*y1; y1:=y1*y2; y2:=y2*y3;
      s:=f(x,y)+s+y
    end;
        ...
end.
```

This program contains designator of function f in the cycle body. We will pass the SRRs constructed into the body of this function. In the result of SRRs construction in the body of f variables u and v will be connect with systems of recurrent relations of (1) type:

$$\begin{aligned} u &= \{0,*,1/2,*,e^{-2h^3+6h^2-24h}e^{h^2/4-3h}, \\ &\quad *,e^{-12h^3+12h^2}e^{h^2/2},e^{-12h^3}\} \\ v &= \{0,*,1,*,e^{2h^2}/e^{-2h^3+6h^2-24h}, \\ &\quad *,e^{4h^2}/e^{-12h^3+12h^2},1/e^{-12h^3}\} \end{aligned}$$

New function in program below evaluates the values u and v using these SRRs and the side effect (variables u and v become global in this variant of program):

```
 function f_new(x,y: real): real;
  { new function that we put in program}
  { instead of f }
  begin
   f_new:=u/(v+1);
   u:=u*u1; u1:=u1*u2; u2:=u2*u3;
   v:=v*v1; v1:=v1*v2; v2:=v2*v3;
  end;

begin
        ...
  h:=b/n; s:=0;

  { Initialization of computations using SRRs }
  y:=1; y1:=exp(-h*h*h+3*h*h-12*h);
  y2:=exp(-6*h*h*h+6*h*h); y3:=exp(-6*h*h*h);
  x:=0; x1:=h;

  { Initialization of computations}
  {    using new obtained SRRs    }
  u:=1/2; u1:=y1*y1*exp(h*h/4-3*h);
  u2:=y2*y2*exp(h*h/2); u3:=y3*y3;
  v:=1; v1:=exp(2*h*h)/(y1*y1);
  v2:=exp(4*h*h)/(y2*y2); v3:=1/u3;

  s:=f_new(x,y)+s+y;
  { The resulting cycle }
  for i:=1 to n do
    begin
      x:=x+x1;
      y:=y*y1; y1:=y1*y2; y2:=y2*y3;
      s:=f_new(x,y)+s+y
    end;
        ...
end.
```

Let b=0.5 and n=10000. The time for performing of the first, second and third variants of a program is 133.08, 105.40 and 58.98 (in seconds). Examples were executed in Turbo-Pascal 6.0 on the IBM PS/2 (model 50).

Such technique of procedure's optimization has following visible defect: if function f is called from different places of a cycle or from different cycles, the definition of f must be manifold in sufficient quantity of copies with different names. But the main part of computations will be removed from these definitions in the result of optimization. After that the SCOPE's facilities can be applied to the assignments that appear in front of the cycles.

4 Conclusion

Computer algebra provides good tools for code optimization. Especially it concerns to "source-to-source" optimization. But existent tools (SCOPE, Gentran, etc.) provide code transmission from computer algebra system to numeric system only. That's why we have started developing in MSU a source-to-source optimization library using Reduce as intellectual tool. This library contains algorithms above-mentioned and special tools that provide reliable bilateral connection between Reduce and systems for numeric computations on MS DOS computers (Turbo-Pascal, Turbo-C, MathCad, etc.)

References

1. *van Hulzen J.A., Hulsof B.J.A., Gates B.L. and van Heerwaarden M.C.* "A code optimization package for REDUCE", Proceeding ISSAC'89 (G.Gonnet, ed.), 163–170, New-York, ACM Press, 1989.

2. *Geddes K.O., Fee G.J.* "Hybrid Symbolic-Numeric Integration in Maple", Proceeding ISSAC'92 (Paul S. Wang, ed.), 36–41, New-York, ACM Press, 1992.

3. *Paul S. Wang* "A System Independent Graphing Package for Mathematical Function", Proceeding DISCO'90, LNCS, N 429, 245–254, 1990.

4. *Gates, B.L.*: "GENTRAN: An automatic code generation facility for REDUCE", A.C.M. SIGSAM Bulletin 19, 3, 24–42. New York: A.C.M. (1985).

5. *Char B.W., Geddes K.O., Gonnet G.H., Leong B., Monagan M.B., Watt S.M* Maple 5. Language Reference Manual, Springer-Verlag, 1991.

6. *Grant O. Cook,Jr.* "Code Generation in ALPAL using Symbolic Techiques", Proceeding ISSAC'92 (Paul S. Wang, ed.), 27–35, New-York, ACM Press, 1992.

7. *Aho A.V., Ullman J.D.* The theory of parsing, translation and compiling. Vol. 2: Compiling. (Prentice-Hall, Inc. Englewood Cliffs, N.J. 1973)

8. *Zima E.V.* "Automatic Construction of Systems of Recurrence Relations". Journal of Computational Mathematics and Mathematical Physics., Vol.24, N 6, (1984), pp. 193–197.

9. *Zima E.V.* "Construction of Economical Computation Formulae in Computer Algebra Systems." Proc. 4th Intern. Conference on Computer Algebra in Phisical Research,(CAPR), Singapore: World Scientific, 1991, pp.112–116.

10. *Char B.W., Geddes K.O., Gonnet G.H., Leong B., Monagan M.B., Watt S.M* Maple 5. Library Reference Manual, Springer-Verlag, 1991.

Appendix

The time (in seconds) for performing of the sections A–E in Reduce 3.3 on IBM PS/2 (model 50) is presented in the table below. Column B presents the fastest time for computations needed that received using SCOPE facilities. Columns D and E present the fastest variant of the same computations that received using SRR's technique and SCOPE facilities. Column E contains the time of initial assignments' evaluation and D — the time of main cycle's performing. Values of needed sum that received using B section and (D+E) sections are differ only in the last digit of mantissa (this difference is 10^{-18} for $n = 40$).

n	A	B	C	D	E
10	58.44	58.06	26.03	6.87	11.43
20	120.62	119.3	26.03	14.12	11.43
30	180.43	177.9	26.03	22.41	11.43
40	245.9	240.63	26.03	31.36	11.43
50	310.77	302.69	26.03	36.74	11.43

The following table presents the same information that received in Reduce 3.3 on 486DX-50 based computer.

n	A	B	C	D	E
10	7.25	6.98	3.19	1.16	1.48
20	14.44	14.72	3.19	2.41	1.48
30	22.08	21.8	3.19	3.57	1.48
40	30.38	29.66	3.19	4.78	1.48
50	36.69	36.25	3.19	6.43	1.48