AN EXPERIMENTAL APPRAISAL OF ADA CONCURRENCY*

Shohei Fujita

Department of Computer Science
Tokyo Institute of Technology

EXTENDED ABSTRACT

Control abstraction mechanisms play a crucial role to create reliable and high-quality software, particularly in the field of embedded computer systems. Control structure - the mechanisms by which programmers can specify the flow of execution among the components of a program - can be classified as

(1) statement level control structures
-- to order the activations of individual statements

(2) program unit level control structures
-- to order the activations of program units

(3) processing unit level control structures
-- to order the activations of processing units.

In past programmers writing program for embedded computer systems have been required to enable concurrent execution of their programs by interacting with operating systems [1]. Ada provides concurrency at program unit level and even at processing unit level**, as a feature of the language, to be used by the programmers at the application level.

An Ada concurrent system is composed of a set of tasks whose executions can be (conceptually) overlapped in time,i.e., the start of a task can occure when the previously executing task is not terminated.

The tasks in Ada concurrent system must cooperate in order to achieve a common goal. For example, in the producer-consumer problem, correct cooperation requires [4]

A principle of concurrent system: partial ordering among actions

$$P_k \rightarrow C_k \quad \text{and} \quad C_k \rightarrow P_{k+N} \quad \text{for all } k \quad (*)$$

where $P_k(C_k)$ denotes the production (consumption) of kth element, and "→" should be read as "precedes".

This paper presents an experimental appraisal of Ada concurrent system for the principle (*) by using concurrent numerical algorithm: ACN [2]. The Ada system espoused here is MicroAda/SuperMicro produced by Western Digital Corporation.

CONCURRENT PROGRAM

A sample of Ada concurrent program for ACN algorithm is shown in Fig. 1. The system of nonlinear equations used is

$$f_1(x) = x_1^2 + x_2^2 - 1$$

$$f_2(x) = x_1 - x_2^2.$$

REFERENCES

[1] Downes, V.A. & S.J. Goldsack : Programming Embedded Systems with Ada. PHI, (1982).

[2] Fujita, S. : "Distributed MIMD multiprocessor system with MicroAda/Super-Micro(TM) for asynchronous concurrent Newton's algorithms," Proc. 5th ACM-SIGSMALL Symp. pp. 49 - 59, (1982).

[3] Fujita, S. : "Asynchronous algorithm and programming for decentralized computing systems: A pragmatic example," 6th Int. Conf. Software Engineering : Poster Session, D2-22, (1982).

[4] Ghezzi, C. & M. Jazayeri : Programming Language Concepts. John Wiley & Sons, (1982).

[5] Hoare, C.A.R. : Private discussion, (Sep. 1982).

[6] U.S. Department of Defense : "Ada 82: Reference manual for the Ada programming language," Draft Proposed ANSI Standard Document, (1982).

ACKNOWLEDGEMENTS

The author would like to thank Prof. T. Fukao for his support. The discussions and cooperations with the members of research group: H. Ohkata, H. Ohno, T. Ohno, O. Iwasaki, T. Kyozuka, and A. Ohshima are gratefully acknowledged.

* This work was supported in part by the Ministry of Education, Japanese Government, under Grant 56460104.

Ada is a trademark of the U.S.Department of Defense (Ada Joint Program Office).

** Ada distributed system will be discussed elsewhere.

```
-- Program Title :
--  ACN(Asynchronous Concurrent Newton)

-- Facility :
--  This program solves a system of nonlinear equations.

-- Feature :
--  The order among tasks is free!

with text_io,input_output;

procedure ACN is

  use text_io,input_output;

  MAXSIZE:constant INTEGER:=2;
  subtype INDEX is INTEGER range 1..MAXSIZE;
  type RMATRIX is array(INDEX,INDEX) of FLOAT;
  type RARRAY is array (INDEX) of FLOAT;
  type IARRAY is array(INDEX) of INTEGER;

  task LINEAR;

  task JACOBIAN is
     entry STOP;
  end JACOBIAN;

  task BUFFER_J is
     entry SEND (J:in RMATRIX);
     entry RECEIVE (J:out RMATRIX);
  end BUFFER_J;

  task BUFFER_L is
     entry SEND (X:in RARRAY);
     entry RECEIVE (X:out RARRAY);
  end BUFFER_L;

  -----------------------------------------------------------------

  task body LINEAR is        -- This task solves linear systems.
     J:RMATRIX;
     X,ROOT,B,C:RARRAY;
     NORM,BUF,BUF1,BUF2,PRECIS:FLOAT;
     MAX_ITERATION,I:INTEGER;
     REPEAT,FOUND:BOOLEAN;
     ch:character;
     NAME:STRING(1..20);
     SUB:IARRAY;
     SINGULAR:exception;

    package IN_OUT is       -- This package treats input/output.
      procedure INPUT;
      procedure OUTPUT;
      procedure MESSAGE;
    end IN_OUT;

   ----------------------------------------------------------

   package body IN_OUT is
     prin : out_file;
     cons : in_file;

     procedure INPUT is
       begin
         put("Output file? : ");
         get(NAME);
         open(prin,NAME);
         open(cons,NAME);
```

```
     put_line(prin,"");put_line(prin,"Initial value is");
     for I in 1..2 loop
       put(prin,"  X(");put(prin,I);put(prin,")=");
       get(cons,BUF);put(prin,BUF);X(I):=BUF;
       put_line(prin,"");
     end loop;
     put(prin,"Precise ");
     get(cons,PRECIS);put(prin,PRECIS);
     put_line(prin,"");
     put(prin,"Enter max number of iteration :  ");
     get(cons,MAX_ITERATION);
     put_line(prin,"");
   end INPUT;

   procedure OUTPUT is
      begin
        if FOUND
        then put_line(prin,"The solution is");
             for I in 1..2 loop
                put(prin,"   X(");put(prin,I);put(prin,")=");
                BUF:=ROOT(I);put(prin,BUF);put_line(prin,"");
             end loop;
        else put(prin,"The solution was not found in ");
             put(prin,MAX_ITERATION);
             put_line(prin," iterations");
             put_line(prin,"The final value is");
             for I in 1..2 loop
                put(prin,"   X(");put(prin,I);put(prin,")=");
                BUF:=ROOT(I);put(prin,BUF);put_line(prin,"");
             end loop;
        end if;
        put_line(prin,"continue? ");
        get(cons,ch);
        if (ch = 'y') or (ch = 'Y') then
          REPEAT := true;
        else
          REPEAT := false;
        end if;
        put_line(prin,"");
        put_line(prin,"");
        close(prin);
        close(cons);
     end OUTPUT;

  procedure MESSAGE is
    begin
      put_line(prin,"");put_line(prin,"");
      put_line(prin,"The matrix of coefficient  is singular");
      put_line(prin,"");
      put_line(prin,"Cannot continue processing");
      put_line(prin,"");
    end MESSAGE;

  end IN_OUT;

------------------------------------------------------------

function F1(X:in RARRAY) return FLOAT is
   begin
      return X(1)**2+X(2)**2-1.0;
   end F1;

function F2(X:in RARRAY) return FLOAT is
   begin
      return X(1)-X(2)**2;
   end F2;
```

```
        procedure LU_FACTOR(N:in INTEGER;A:in out RMATRIX) is

         use IN_OUT;

         INDEX,J:INTEGER;
         PIVOT,MAX,AB,T:FLOAT;

         begin
             for I in 1..N loop
               SUB(I):=I;
             end loop;
             for K in 1..N-1 loop
               MAX:=0.0;
               for I in K..N loop
                 T:=A(SUB(I),K);
                 AB:=abs(T);
                 if AB>MAX then
                   MAX:=AB;
                   INDEX:=I;
                 end if;
               end loop;
               if MAX<=1.0E-7 then
                 raise SINGULAR;
               end if;
               J:=SUB(K);
               SUB(K):=SUB(INDEX);
               SUB(INDEX):=J;
               PIVOT:=A(SUB(K),K);
               for I in K+1..N loop
                 A(SUB(I),K):=-A(SUB(I),K)/PIVOT;
                 for J in K+1..N loop
                   A(SUB(I),J):=A(SUB(I),J)+A(SUB(I),K)*A(SUB(K),J);
                 end loop;
               end loop;
             end loop;
             for I in 1..N loop
               BUF:=A(SUB(I),I);
               if abs(BUF)<1.0E-7 then
                 raise SINGULAR;
               end if;
             end loop;
           exception when SINGULAR=>
             MESSAGE;  -- Message for unsolvable systems.
             raise;
         end LU_FACTOR;

       procedure SOLVE(N:in INTEGER;A:in RMATRIX;
                        C:in RARRAY;X:out RARRAY) is

         begin
           if N=1 then
             X(1):=C(1)/A(1,1);
           else
             X(1):=C(SUB(1));
             for K in 2..N loop
               X(K):=C(SUB(K));
               for I in 1..K-1 loop
                 X(K):=X(K)+A(SUB(K),I)*X(I);
               end loop;
             end loop;
             X(N):=X(N)/A(SUB(N),N);
             for K in reverse 1..N-1 loop
               for I in K+1..N loop
                 X(K):=X(K)-A(SUB(K),I)*X(I);
               end loop;
               X(K):=X(K)/A(SUB(K),K);
             end loop;
           end if;
         end SOLVE;
```

```
  use IN_OUT;

 begin
  loop
    INPUT;
    FOUND:=false;
    I:=1;
    while not FOUND and (I<=MAX_ITERATION) loop
       BUFFER_L.SEND(X);  --  entry call
       BUFFER_J.RECEIVE(J);  --  entry call
       LU_FACTOR(2,J);
       B(1):=-F1(X);
       B(2):=-F2(X);
       SOLVE(2,J,B,C);
       for K in 1..2 loop
         ROOT(K):=X(K)+C(K);
       end loop;
       BUF1:=F1(ROOT);
       BUF2:=F2(ROOT);
       NORM:=abs(BUF1)+abs(BUF2);
       if NORM<PRECIS
          then FOUND:=true;
       end if;
       X:=ROOT;
       I:=I+1;
    end loop;
    OUTPUT;
    if not REPEAT then
      exit;
    end if;
  end loop;
  JACOBIAN.STOP;  --  entry call

  exception
    when SINGULAR =>
             JACOBIAN.STOP;  --  entry call
end LINEAR;

------------------------------------------------------------

task body BUFFER_L is
   SIZE:constant INTEGER :=5;
   BUFFER:array(1..SIZE) of RARRAY;
   NEXTIN,NEXTOUT:INTEGER range 1..SIZE :=1;
   CONTAINS:INTEGER range 0..SIZE :=0;
begin
    loop
      select
         when CONTAINS>0 =>
            accept RECEIVE(X:out RARRAY) do
               X:=BUFFER(NEXTOUT);
            end RECEIVE;
            NEXTOUT:=NEXTOUT mod SIZE+1;
            CONTAINS:=CONTAINS-1;
      or
         when CONTAINS<SIZE =>
            accept SEND(X:in RARRAY) do
               BUFFER(NEXTIN):=X;
            end SEND;
            NEXTIN:=NEXTIN mod SIZE +1;
            CONTAINS:=CONTAINS+1;
      or
        terminate;
      end select;
   end loop;
end BUFFER_L;

------------------------------------------------------
```

```
task body JACOBIAN is      --  This task computes Jacobian matrix.
   X:RARRAY;
   J:RMATRIX;
begin
   loop
      select
         accept STOP do
            exit;
         end STOP;
      else              -- Conditional entry call
         BUFFER_L.RECEIVE(X);  --  entry call
         J(1,1):=2.0*X(1);
         J(1,2):=2.0*X(2);
         J(2,1):=1.0;
         J(2,2):=-2.0*X(2);
         BUFFER_J.SEND(J);  --  entry call
      end select;
   end loop;
end JACOBIAN;

----------------------------------------------------

task body BUFFER_J is
   SIZE:constant INTEGER :=5;
   BUFFER:array (1..SIZE) of RMATRIX;
   NEXTIN,NEXTOUT:INTEGER range 1..SIZE :=1;
   CONTAINS:INTEGER range 0..SIZE :=0;
begin
    loop
      select
         when CONTAINS>0 =>
            accept RECEIVE(J:out RMATRIX) do
               J:=BUFFER(NEXTOUT);
            end RECEIVE;
            NEXTOUT:=NEXTOUT mod SIZE+1;
            CONTAINS:=CONTAINS-1;
      or
         when CONTAINS<SIZE =>
            accept SEND(J:in RMATRIX) do
               BUFFER(NEXTIN):=J;
            end SEND;
            NEXTIN:=NEXTIN mod SIZE+1;
            CONTAINS:=CONTAINS+1;
      or
        terminate;
      end select;
   end loop;
end BUFFER_J;

----------------------------------------------------------

begin
  null;
end ACN;
```

Fig.1 Ada Concurrent Program