Experimental Performance Analysis of the Ada95 and Java Parallel Program on SMP Systems

Dmitry Korochkin
National Technical University of Ukraine –
Kiev Polytechnic Institute
Prospect Peremogy 37, 03056,
Kiev, Ukraine
cora@comsys.ntu-kpi.kiev.ua

Sergey Korochkin
National Technical University of Ukraine –
Kiev Polytechnic Institute
Prospect Peremogy 37, 03056,
Kiev, Ukraine
cora@comsys.ntu-kpi.kiev.ua

ABSTRACT

In this paper, we describe the results of experimental performance analysis of Ada and Java parallel program on SMP systems. Speed up of Ada and Java set programs to solving some linear algebra problems are presented.

Categories and Subject Description

D.1.3 Concurrent Programming –*Parallel programming*
D.2.8 Metrics- P*erformance measures*
D.3.3 Language Constructs and Features – *Concurrent programming structures*
D.4.1 Process Management
C.4 PERFORMANCE OF SYSTEMS

General Terms: Performance

Keywords: Ada, Java, SMP systems, task, thread, mutual exclusion, synchronization.

1. INTRODUCTION

The Symmetrical Multiprocessors Systems (SMP) provide high performance for many applications. But there are problems efficient parallel programs creation. Compared to sequential programming in parallel programming there are many specific problems, which need a special language and system facilities for theirs solving.

There are a two ways of parallel program creation– using of libraries (Win32, OpenMP, MPI, PVM) or using of parallel programming languages (Java, Ada). Both Ada and Java support programming for SMP systems [1,2]. A comparison of the concurrency features of Ada95 and Java is presented in [3].

Our main goals are in development and practical test of Ada and Java parallel programs on the real SMP system. We intend to research an influence of different mechanism processes and different facilities of process communication on run-time Ada and Java parallel programs.

2. PROGRAMMING FOR SMP SYSTEMS

SMP is a class of parallel computer systems, based on shared memory. A simply 2-4 processors SMP systems today are used as a workstation, servers and in nodes of distributed claster systems.

Programming for SMP systems is based on the work with processes and includes solving of some special problems. There are two main problems: *mutual exclusion* for access to shared resources and event processes *synchronization*. Both Ada and Java provide the different facilities for work with processes, mutual exclusion and process synchronization.

2.1 Process Declaration

Processes in Java are realized via *threads*, in Ada - via *tasks*. Classes and task types provide a creation of a processes groups. Constructors in Java and discriminants in Ada provide a process parameterization. In Java thread is run by `start()` method. In Ada task is run by automatically.

2.2 Mutual Exclusion

Java supplies a mutual exclusion via volatile variables and via synchronized methods and blocks. Ada provides mutual exclusion via atomic variables (pragmas Atomic and Volatile), semaphores mechanism (package `Ada.Synchronous_Task_Control`), protected object/types.

2.3 Process Synchronous Control

Thread synchronization in Java is provided by methods `wait()`/ `notify()`/ `notifyAll()`. Method `join()` is used for parent/child threads synchronization. Tasks synchronization in Ada is supplied

by semaphores and protected entries on protected object/types.

3. PARALLEL PROGRAMMING

Development of parallel program for SMP system includes few steps: creation of main parallel algorithm, algorithms for each parallel process, choice of facilities for mutual exclusion and process synchronization, programming.

Ada and Java parallel program for SMP systems is a set of concurrent processes (tasks or threads) which communicate via shared resources (mutual exclusion problem) or events (event synchronize problem).

We present the creation steps of parallel program to scalar vector multiplication problem a = (B*C) in SMP system with two processors. This program includes two processes. Input data and output result belong to first process.

Step1. General parallel algorithm
Includes main parallel mathematical algorithm to solving a problem. (h = n/2):

$$a1 = B(1) * C(1) \quad + \ldots + \ B(h) * C(h)$$
$$a2 = B(h+1) * C(h+1) + \ldots + \ B(n) * C(n)$$
$$a \ = a1 + a2$$

Step2. Algorithms of processes
Includes algorithm for each process with critical sections (CS) and points of processes synchronization (PS). The critical section is an access to shared variable (a) when result is formed. There are two problems of processes synchronization Process P2 is waiting for process P1, because of process P1 inputs vectors B and C, (PS1 – input synchronization). Process P1 is waiting for finish of calculation in P2, because of process P1 outputs the full result a (PS2 – output synchronization).

    Process P1
 1. Input B, C
 2. **Signal** for P2   -- (PS1)
 3. Calculation a1
 4. Calculation a = a + a1   -- (CS)
 5. **Wait** for result P1   -- (PS2)
 6. Output result a

    Process P2
 1. **Wait** for input in P2   -- (PS1)
 2. Calculation a2
 3. Calculation a   -- (CS)
 4. **Signal** for P1   -- (PS2)

Step3. Choice of facilities for mutual exclusion in CS and synchronization in PS1 and PS2. Development of process communication structure. Analysis this structure for deadlocks.

Step4. Using algorithms of each process P1 and P2 and structure of process communication the program is created

4. METODOLOGY

All experiments were conducted on a workstation with dual processors (MB: Asus BX, CPU: Petium II – 400, SDRAM: 256MB). Used software: OS Windows NT (2000), Linux, JBuilder, compilers Aonix, Gnat.

Performance analysis is based on comparison of run- time of both Ada and Java program for linear algebra problems. Considered a typical problems linear algebra for vectors (A) and matrix (MA): a = (B*C), A = B*MC , MA=MB*MC for different size of vectors and matrixes, solution of systems of linear algebraic equations (iterative Jacobi method). For these linear algebra problems were created the sequential and parallel Ada and Java program versions.

Ada parallel version had a three realisation. In first realization we used pragmas `Atomic` and `Volatile` for mutual exclusion and semaphores from package `Ada.Synchronous_Task_Control` for processes synchronization. In second realisation we used only semaphores from package `Ada.Synchronous_ Task_Control`. In third realisation we used a protected types for both problems of processes communications.

In parallel Java version we used the synchronized methods for mutual exclusion and `wait()/notify()` methods for thread synchronization. For vector and matrix elements we used a different types: int, long, double, float, integer. We used a vector and matrix size n = 100 – 10**6.

For real SMP system we have got a run-time for each program : To – for sequential program, Tp – for parallel programs. For Ada program there are three Tp: 1- for pragmas and semaphores, 2 – for semaphores, 3 – for protected types. Speed up: Su = To/Tp.

5. PERFORMANCE ANALYSIS

We developed a set Ada and Java programs for solve some problems of linear algebra and executed on dual SMP systems. Sequential program versions (with one task and thread or without processes) tested on SMP systems where second processor was switched off.

The values of speed up are presented in Table 1.

Table 1: Speed up

Operation	N	Speed Up			
		Java	Ada		
			1	2	3
a = (B*C)	10^6	1,13	1,18	1,18	1,34
A = B * MC	1000	1.33	1,45	1,45	1,41
MA=MB*MC	100	1,6	1,9	1,9	1,9
Jacobi method	2000	1,27	-	1,52	1,6

6. CONCLUSIONS

Compared to Java the Ada versions provide in general the best speed up (1,18-1,9 versus 1,13-1,6). Moreover the run-time for Ada program can be optimised by checking suppress.

Ada has more facilities for mutual exclusion and process synchronous control then Java. The versions Ada programs with different facilities of process communication have not big differences in run-time (~5-6%). Pragmas and semaphores are simply in using, protected types provide more ways (entry, function, procedure) for practical solve of different communication problem. But the using of protected types needs for attention by programmer under a development of optimal structure of protected unit. Good result could be get by combination of these facilities.

In general Java in parallel programming provided a dual impression. Java programs surprised a high speed for interpreting language and non-stable run-time influenced from program structure. Ada programs provided more stabile run-time for set tests of one-program then Java programs.

In next experiments we plan to use Aonix compiler for test of Ada program in JVM. Moreover we intend to consider an using applying the Win32 facilities in Ada programs.

Future work will include an experimental performance analysis Ada and Java facilities for applications in real distributed systems.

7. REFERENCES

[1] A. Korochkin . *Ada95: Introduction in Programming* . Svit, Kiev, 1999.

[2] P. Naughton P, H.Schildt *Java2: The Complete Reference*. Osborne McGraw-Hill, 1999.

[3] B. M. Brosgol. A Comparison of the Concurrency and Real-Time Features of Ada95 and Java. *Ada User Journal,* 19 (4): 225-257, January 1999.

APPENDIX

A. PROGRAMS SAMPLES

This Appendix presents the Java and Ada program versions for scalar vector multiplication operation
a = (B*C) for dual SMP system . In Ada program are used the pragma Atomic and mechanism of semaphores, in Java – synchronized methods.

A.1 Java version

```
package Vector;

public class Data {
  int a = 0;
  int N = 1000000;
  int H = N/2;
```

```
  int B[] = new int [N];
  int C[] = new int [N];
  private int FlagIn = 0;
  private int FlagOut = 0;

  public synchronized void
   puta(int x){ a = a + x;}

  public synchronized void waitIn(){
   try{
     if (FlagIn = = 0)
       wait();
   }catch(Exception e) {}
  }
  public synchronized
         void signalIn(){
   notify();
   FlagIn = 1;
  }
  public synchronized void waitOut(){
   try{
     if(FlagOut = = 0)
        wait();
   }catch(Exception e) {}
  }
  public synchronized
        void signalOut(){
   notify();
   FlagOut = 1;
  }
 }
}
public class aThread extends Thread{
  int a1 = 0;
  Data Z;
  public aThread(Data q){
   Z = q;
  }
  public void run(){
   // Input B,C
   for(int i = 0; i< Z.N; i++){
     Z.B[i] = 1;
     Z.C[i] = 1;
   }
   // Signal to P2
   Z.signalIn();      //(PS1)
   // Calculate a1
   for (int i = 0; i < Z.H; i++)
      a1 = a1 + Z.B[i]*Z.C[i];
   //Result
   Z.puta(a1);       // (CS)
   //Wait for P2
   Z.waitOut();      // (PS2)
   //Output result
   Z.geta();
  }
}
public class bThread extends Thread{
  int  a2 = 0;
  Data Z;

  public bThread(Data q) {
```

```
    Z = q;
  }
  public void run(){
   // Wait for P1
   Z.waitIn();        //(PS1)

   // calculate a2
   for (int i= Z.H; i< Z.N; i++)
     a2 = a2 + Z.B[i]*Z.C[i];

   // Result
   Z.puta(a2);        //(CS)

   // Signal for P1
   Z.signalOut();     //(PS2)
  }
}

import Vector.*;
public class aBC_SMP {
  public static void main
        (String [] args){

  Data D = new Data();
  aThread P1 = new aThread(D);
  bThread P2 = new bThread(D);
  P1.start();
  P2.start();
  }
}
```

A.2 Ada version

```
generic
  N: integer;
package Data is
  H : integer:= N/2;
  type Vector is array (Positive
        range<>) of integer;
  subtype VectroN is Vector(1..N);
  subtype VectorH is Vector(1..H);
end Data;

with Data;
with Ada.Synchronous_Task_Control;
use Ada.Synchronous_Task_Control;
procedure aBC_SMP is

  package Data1 is new
          Data(1000000);
  use Data1;

  B, C : VectorN;
  a   : integer := 0;

  -- facilities for synchronization
  pragma atomic(a);

  SemIn, SemOut : Suspend_Object;

  task P1;
  task body P1 is
```

```
  a1: integer:= 0;

begin
  -- input B, C

  for i in 1..N loop
    B(i):= 1;
    C(i):= 1;
  end loop;

  -- Signal for P2
  Set_True(SemIn);    -- (PS1)

  -- Calculation a1
  for i in 1..H loop
    a1:= a1 + B(i)* C(i);
  end loop;


  -- Result
  a:= a + a1;        -- (CS)

  -- Wait for P2      -- (PS2)
  Suspend_Until_True(SemOut)

  -- Output result
 put(a);
end P1;

task P2;

task body P2 is

  a2: integer:= 0;

begin

  --Wait for P1      --(PS1)
  Suspend_Until_True(SemIn);

  -- Calcualation a2
  for i in H+1..N loop
    a2:= a2 + B(i)* C(i);
  end loop;

  -- Result
  a := a + a2;       -- (CS)

  -- Signal for P1
  Set_True(SemOut);   -- (PS2)

  end P2;

begin

  null;

end aBC_SMP;
```

56