

Міністерство освіти і науки України
Національний авіаційний університет
Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютеризованих систем управління

Лабораторна робота № 1.1
з дисципліни «Паралельні і розподілені обчислення»
на тему «Ада. Семафори»

Виконав:
студент ФККПІ
групи СП-425
Клокун В. Д.
Перевірів:
Корочкін О. В.

Київ 2019

1. ЗАВДАННЯ РОБОТИ

Розробити програму для заданої паралельної комп'ютерної системи зі спільною пам'яттю (рис. 1).

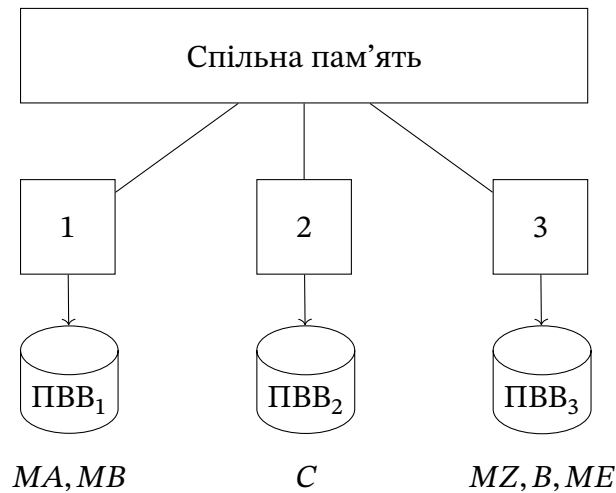


Рис. 1: Задана паралельна комп'ютерна система зі спільною пам'яттю

Програма, розроблена для даної системи, повинна обчислювати значення такого виразу:

$$MA = (MB \cdot MZ) + (B \cdot C) \cdot ME.$$

Розробити програму на мові програмування «Ада», використовуючи для взаємодії потоків (задач) семафори мови «Ада» з пакету `Ada.Synchronous_Task_Control`.

2. ХІД РОБОТИ

2.1. Побудова паралельного алгоритму

Необхідно паралельно обчислити значення виразу $MA = (MB \cdot MZ) + (B \cdot C) \cdot ME$. Для цього складаємо паралельний алгоритм:

1. Позначимо скалярний добуток векторів як $a = B \cdot C$.
2. Обчислимо частини, які складатимуть скалярний добуток векторів:

$$a = B_{H_1} \cdot C_{H_1}, \quad a_i = B_{H_i} \cdot C_{H_i}, \quad a \in \{2, 3\}.$$

3. Обчислимо скалярний добуток векторів, склавши частини:

$$a = a + a_2 + a_3.$$

4. Паралельно обчислюємо матрицю-результат по частинам:

$$MA_{H_i} = (MB_{H_i} \cdot MZ) + a \cdot ME_{H_i}, \quad a \in \{1, 2, 3\}.$$

Спільні ресурси: a, a_2, a_3 — для обчислення скалярного добутку; MZ — для множення матриць.

2.2. Розробка алгоритмів потоків (задач)

Розробивши паралельний алгоритм, переходимо до розробки алгоритмів потоків. Представимо їх у вигляді таблиці.

Табл. 1: Паралельний алгоритм потоку 1

Дія	Точки синхронізації
Ввести MB	
Подати сигнал, що MB введена, в задачу $T2$	$S_{2,MB}$
Подати сигнал, що MB введена, в задачу $T3$	$S_{3,MB}$
Очікувати введення C	$W_{1,C}$
Очікувати введення MZ, B, ME	$W_{1,MZBME}$
Скопіювати $MZ_1 := MZ$	Критична ділянка
Обчислити $a := C_{H_1} \cdot B_{H_1}$	Критична ділянка
Чекати готовності a_2 із задачі $T2$	W_{1,a_2}
Чекати готовності a_3 із задачі $T3$	W_{1,a_3}
Обчислити $a = a + a_2 + a_3$	Критична ділянка
Подати сигнал, що готове значення a , в задачу $T2$	$S_{2,a}$
Подати сигнал, що готове значення a , в задачу $T3$	$S_{3,a}$
Обчислити $MA_{H_1} = (MB_{H_1} \cdot MZ_1) + a \cdot ME_{H_1}$	
Очікувати готовності MA_{H_2} від задачі $T2$	$W_{1,MA_{H_2}}$
Очікувати готовності MA_{H_3} від задачі $T3$	$W_{1,MA_{H_3}}$
Вивести MA	

Табл. 2: Паралельний алгоритм потоку 2

Дія	Точки синхронізації
Ввести C	
Подати сигнал, що C введена, в задачу $T1$	$S_{1,C}$
Подати сигнал, що C введена, в задачу $T3$	$S_{3,C}$
Очікувати введення MZ, B, ME	$W_{2,MZBME}$
Скопіювати $MZ_2 := MZ$	Критична ділянка
Обчислити $a_2 := C_{H_2} \cdot B_{H_2}$	Критична ділянка

Табл. 2: Паралельний алгоритм потоку 2

Дія	Точки синхронізації
Подати сигнал, що готове значення a_2 , в задачу $T1$	S_{1,a_2}
Очікувати готовності a від задачі $T1$	$W_{2,a}$
Скопіювати значення $ac_2 := a$	Критична ділянка
Очікувати введення MB	
Обчислити $MA_{H_2} = (MB_{H_2} \cdot MZ_{H_2}) + a \cdot ME_{H_2}$	$W_{2,MB}$
Дати сигнал, що готове значення MA_{H_2} , задачі $T1$	$S_{1,MA_{H_2}}$

Табл. 3: Паралельний алгоритм потоку 3

Дія	Точки синхронізації
Ввести MZ, B, ME	
Подати сигнал, що MZ, B, ME введені, в задачу $T1$	$S_{1,MZBME}$
Подати сигнал, що MZ, B, ME введені, в задачу $T2$	$S_{2,MZBME}$
Очікувати введення C	$W_{3,C}$
Обчислити $a_3 := C_{H_3} \cdot B_{H_3}$	Критична ділянка
Подати сигнал, що готове значення a_3 , в задачу $T1$	
Очікувати готовності a від задачі $T1$	S_{1,a_3}
Скопіювати значення $ac_3 := a$	$W_{3,a}$
Очікувати введення MB	Критична ділянка
Обчислити $MA_{H_3} = (MB_{H_3} \cdot MZ_{H_3}) + a \cdot ME_{H_3}$	
Дати сигнал, що готове значення MA_{H_3} , задачі $T1$	$W_{3,MB}$
	$S_{1,MA_{H_3}}$

2.3. Розробка структурної схеми взаємодії задач

Розроблюємо структурну схему взаємодії задач (рис. 2). Розробивши схему, вкажемо, навіщо призначені семафори, які в ній використовуються:

Sem_{cr}	Призначений для доступу до спільних ресурсів.
Sem_{2,MB}	Призначений для синхронізації задачі $T2$ після введення MB .
Sem_{3,MB}	Призначений для синхронізації задачі $T3$ після введення MB .
Sem_{2,a}	Призначений для синхронізації задачі $T2$ після обчислення a .
Sem_{3,a}	Призначений для синхронізації задачі $T3$ після обчислення a .
Sem_{1,C}	Призначений для синхронізації задачі $T1$ після введення C .
Sem_{3,C}	Призначений для синхронізації задачі $T3$ після введення C .
Sem_{1,a_2}	Призначений для синхронізації задачі $T1$ після обчислення a_2 .
Sem_{1,MA_{H_2}}	Призначений для синхронізації задачі $T1$ після обчислення MA_{H_2} .

Sem _{1,MZBME}	Призначений для синхронізації задачі $T1$ після введення MZ , B , ME .
Sem _{2,MZBME}	Призначений для синхронізації задачі $T2$ після введення MZ , B , ME .
Sem _{1,a₃}	Призначений для синхронізації задачі $T1$ після обчислення a_3 .
Sem _{1,MA_{H₃}}	Призначений для синхронізації задачі $T3$ після обчислення MA_{H_2} .

Після розробки структурної схеми можна переходити до розробки програми.

2.4. Розробка програми

Коли структурна схема розроблена, створюємо програму на мові програмування «Ада» (лістинг А.1). Для синхронізації задач використаємо семафори, які надає пакет `Ada.Synchronous_Task_Control`. Після розробки програми запускаємо її на виконання і спостерігаємо результат (рис. 3).

Як видно, програма коректно обчислює значення заданого виразу з вхідними значеннями, заданими у програмі.

3. ВИСНОВОК

Виконуючи дану лабораторну роботу, ми розробили програму для заданої паралельної комп'ютерної системи зі спільною пам'яттю, ознайомились із процесом розробки паралельних алгоритмів, а також із семафорами у мові програмування «Ада».

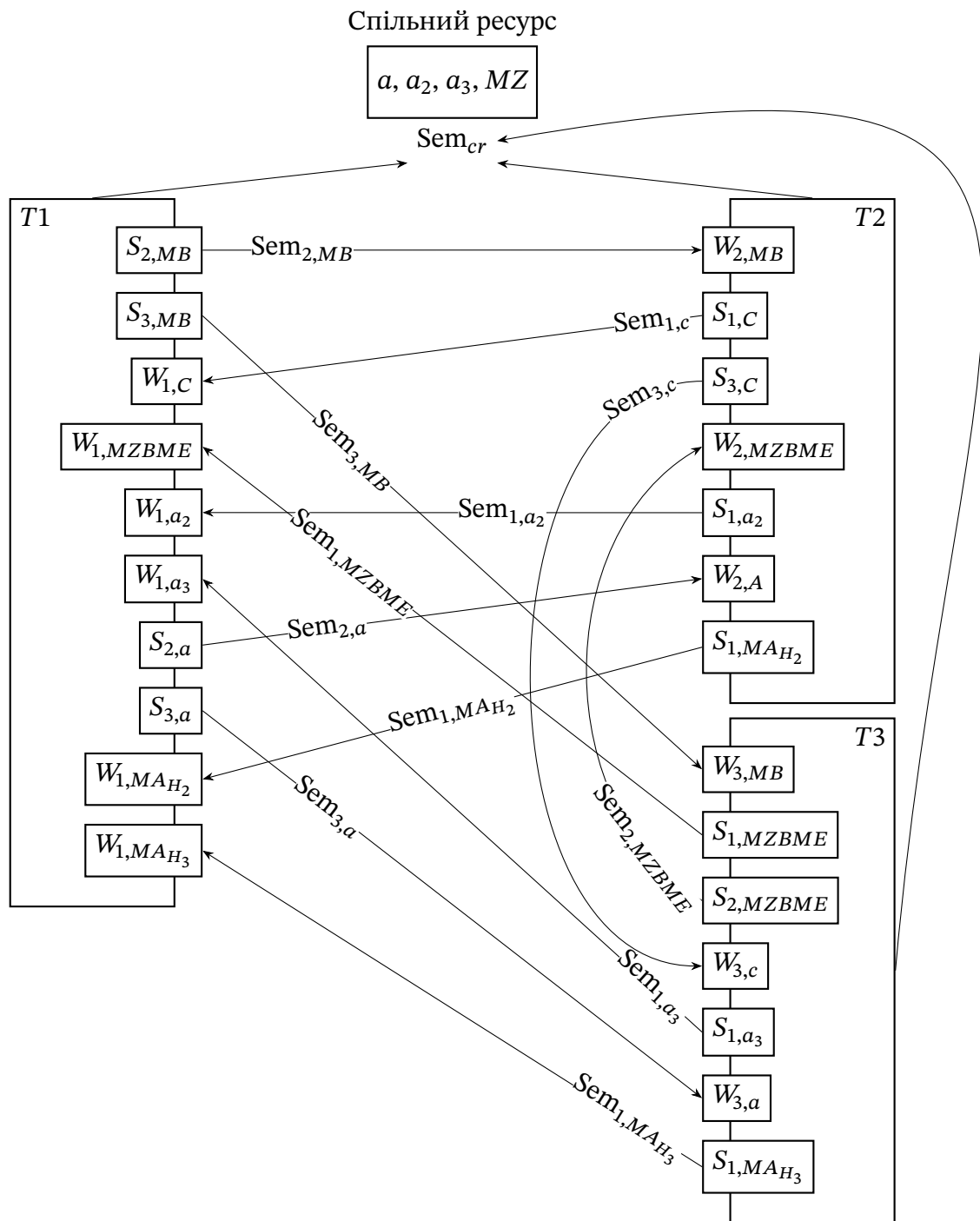
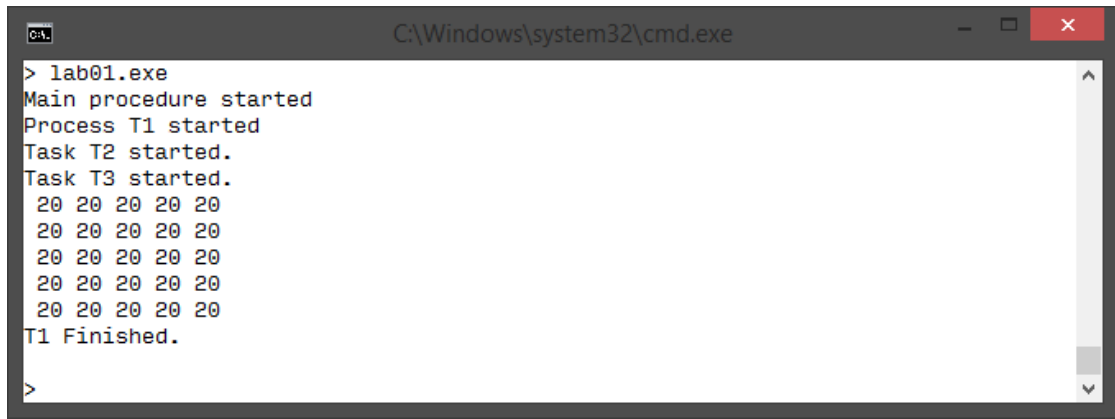


Рис. 2: Структурна схема взаємодії задач



```
C:\Windows\system32\cmd.exe
> lab01.exe
Main procedure started
Process T1 started
Task T2 started.
Task T3 started.
20 20 20 20 20
20 20 20 20 20
20 20 20 20 20
20 20 20 20 20
20 20 20 20 20
T1 Finished.
>
```

Рис. 3: Результат виконання розробленої програми

A. ПРОГРАМА ДЛЯ РОЗВ’ЯЗКУ ПОСТАВЛЕНОЇ ЗАДАЧІ

Лістинг A.1: Початковий код програмного модуля для розв’язання задачі

```
1  -- Parallel and Distributed Computing
2  -- Lab 01: Semaphores
3  -- By Vlad Klokun of SP-425
4  -- Compiled with GNAT Community (20190517-83)
5  --
6  With Ada.Integer_Text_IO, Ada.Text_IO, Ada.Synchronous_Task_Control;
7  Use Ada.Integer_Text_IO, Ada.Text_IO, Ada.Synchronous_Task_Control;
8
9  procedure Lab01 is
10
11      N: integer := 5; -- vector and matrix dimensions
12
13      type Vector is
14          array (Positive range <>) of integer;
15      type Matrix is
16          array (Positive range <>, Positive range <>) of integer;
17
18      P: integer := 3; -- processor count
19      H: integer := N/P; -- vector chunk size
20
21      -- Global variables
22      MA: Matrix(1 .. N, 1 .. N);
23      MB: Matrix(1 .. N, 1 .. N);
24      C : Vector(1 .. N);
25      MZ: Matrix(1 .. N, 1 .. N);
26      B : Vector(1 .. N);
27      ME: Matrix(1 .. N, 1 .. N);
```

```

28      -- Shared resources
29      a, a2, a3: integer;
30
31      -- Semaphores
32      -- Comment format:
33      -- <Destination Thread>: <Explanation>
34      Scr      : Suspension_Object; -- Crit. region
35      S1mzbme: Suspension_Object; -- T1: MZ, B, ME input
36      S2mzbme: Suspension_Object; -- T2: MZ, B, ME input
37      S2mb     : Suspension_Object; -- T2: MB input
38      S3mb     : Suspension_Object; -- T3: MB input
39      S1c      : Suspension_Object; -- T1: C input
40      S3c      : Suspension_Object; -- T3: C input
41      S1a2     : Suspension_Object; -- T1: a2 done
42      S1a3     : Suspension_Object; -- T1: a3 done
43      S2a      : Suspension_Object; -- T2: a done
44      S3a      : Suspension_Object; -- T3: a done
45      S1mah2   : Suspension_Object; -- T1: MA_H2 done
46      S1mah3   : Suspension_Object; -- T1: MA_H3 Done
47
48      procedure InitVector(
49          A: out Vector;
50          size: Integer;
51          val: Integer)
52      is
53      begin
54          for i in 1 .. size loop
55              A(i) := val;
56          end loop;
57      end InitVector;
58
59      procedure PrintVector(
60          A: Vector;
61          size: Integer)
62      is
63      begin
64          for i in 1 .. size loop
65              Put(A(i), 3);
66          end loop;
67      end PrintVector;
68
69      function VectorMul(
70          A: Vector;
71          B: Vector;
72          start: Integer;
73          stop: Integer)
74          return Integer

```



```

75     is
76         res: Integer;
77     begin
78         res := 0;
79         for i in start .. stop loop
80             res := res + A(i) * B(i);
81         end loop;
82         return res;
83     end VectorMul;
84
85     procedure InitMatrix
86         (A: out Matrix;
87          m: Integer;
88          n: Integer;
89          val: Integer)
90     is
91     begin
92         for i in 1 .. m loop
93             for j in 1 .. n loop
94                 A(i, j) := val;
95             end loop;
96         end loop;
97     end InitMatrix;
98
99     procedure PrintMatrix(
100         A: Matrix;
101         m: Integer;
102         n: Integer)
103     is
104     begin
105         for i in 1 .. m loop
106             for j in 1 .. n loop
107                 Put(A(i, j), 3);
108             end loop;
109             New_Line;
110         end loop;
111     end PrintMatrix;
112
113     procedure MMMul(
114         A: Matrix;
115         B: Matrix;
116         C: in out Matrix;
117         m: Integer;
118         n: Integer;
119         p: Integer;
120         start_i: Integer;
121         start_j: Integer

```

```

122     )
123     is
124     begin
125         InitMatrix(C, N, N, 0);
126         for i in start_i .. m loop
127             for j in start_j .. p loop
128                 for k in 1 .. n loop
129                     C(i,j) := C(i,j) + A(i,k) * B(k,j);
130                 end loop;
131             end loop;
132         end loop;
133     end MMMul;
134
135     procedure MMAAdd(
136         A: Matrix;
137         B: Matrix;
138         C: in out Matrix;
139         m: Integer;
140         n: Integer;
141         start_i: Integer;
142         start_j: Integer
143     )
144     is
145     begin
146         for i in start_i .. m loop
147             for j in start_j .. n loop
148                 C(i,j) := A(i,j) + B(i,j);
149             end loop;
150         end loop;
151     end MMAAdd;
152
153     procedure SMMul(
154         A: Integer;
155         B: Matrix;
156         C: in out Matrix;
157         m: Integer;
158         n: Integer;
159         start_i: Integer;
160         start_j: Integer
161     )
162     is
163     begin
164         InitMatrix(C, m, p, 0);
165         for i in start_i .. m loop
166             for j in start_j .. n loop
167                 C(i,j) := A * B(i,j);
168                 null;

```

```

169         end loop;
170     end loop;
171 end SMMul;
172
173 procedure Run_Tasks is task T1;
174
175     task body T1
176     is
177         C1: Vector(1 .. N);
178         MZ1: Matrix(1 .. N, 1 .. N);
179         MT1: Matrix(1 .. N, 1 .. N);
180         MT2: Matrix(1 .. N, 1 .. N);
181     begin
182         put_line("Process T1 started");
183
184         -- Initialize variables but do not consider them ready
185         InitVector(B, N, 0);
186         InitVector(C, N, 0);
187         InitMatrix(MA, N, N, 0);
188         InitMatrix(MB, N, N, 3);
189         InitMatrix(MZ, N, N, 0);
190         InitMatrix(ME, N, N, 0);
191
192         -- MB Initialized
193         Set_True(S2mb);
194         Set_True(S3mb);
195
196         -- Wait and Copy C
197         Suspend_Until_True(S1c);
198         Suspend_Until_True(Scr);
199         Set_False(Scr);
200         C1 := C;
201         Set_True(Scr);
202
203         -- Wait for MZ, B, Me
204         Suspend_Until_True(S1mzbme);
205
206         -- Copy B and MZ
207         Suspend_Until_True(Scr);
208         Set_False(Scr);
209         MZ1 := MZ;
210         Set_True(Scr);
211
212         -- Compute first value of A
213         Suspend_Until_True(Scr);
214         Set_False(Scr);
215         a := VectorMul(C, B, 1, 2);

```

```

216      Set_True(Scr);
217
218      -- Wait for a_2 and a_3
219      Suspend_Until_True(S1a2);
220      Suspend_Until_True(S1a3);
221
222      -- When all parts are ready, put a together
223      Suspend_Until_True(Scr);
224      Set_False(Scr);
225      a := a + a2 + a3;
226      Set_True(Scr);
227      -- Signal other threads
228      Set_True(S2a);
229      Set_True(S3a);
230
231      -- Compute MA_{H}
232      -- MT1 = MB * MZ
233      MMMul(
234          A => MB,
235          B => MZ1,
236          C => MT1,
237          m => 2,
238          n => N,
239          p => N,
240          start_i => 1,
241          start_j => 1
242      );
243
244      -- MT2 = a * ME
245      SMMul(
246          A => a,
247          B => ME,
248          C => MT2,
249          m => 2,
250          n => N,
251          start_i => 1,
252          start_j => 1
253      );
254
255      -- MA = MT1 + MT2 = MB * MZ + a * ME
256      MMAdd(
257          A => MT1,
258          B => MT2,
259          C => MA,
260          m => 2,
261          n => N,
262          start_i => 1,

```

```

263         start_j => 1
264     );
265
266     -- MA_{H_{1}} is done, wait for others
267     Suspend_Until_True(S1mah2);
268     Suspend_Until_True(S1mah3);
269
270     -- When MA_{H_{i}}, where i \in {1, 2, 3} are done too,
271     -- print A.
272     PrintMatrix(MA, N, N);
273     Put_Line("T1 Finished.");
274 end T1;
275
276 task T2;
277 task body T2
278 is
279     -- Local copies
280     ac2: Integer;
281     MZ2: Matrix(1..N, 1..N);
282     -- Temporary values for computing MA_H
283     MT1: Matrix(1..N, 1..N);
284     MT2: Matrix(1..N, 1..N);
285 begin
286     Put_Line("Task T2 started.");
287     -- Input C
288     InitVector(C, N, 1);
289     -- Signal other tasks that C is ready
290     Set_True(S1c);
291     Set_True(S3c);
292
293     Suspend_Until_True(S2mzbme);
294
295     -- Copy MZ2 and compute a2 in one critical section
296     Suspend_Until_True(Scr);
297     Set_False(Scr);
298     MZ2 := MZ;
299     a2 := VectorMul(B, C, 3, 4);
300     Set_True(Scr);
301     -- Signal that a2 is ready
302     Set_True(S1a2);
303
304     -- Wait until a is ready
305     Suspend_Until_True(S2a);
306
307     -- Copy a into this thread. Critical region.
308     Suspend_Until_True(Scr);
309     Set_False(Scr);

```

```

310         ac2 := a;
311         Set_True(Scr);
312
313         -- Wait until MB is ready
314         Suspend_Until_True(S2mb);
315
316         -- Compute  $MA_{\{H\}}$ 
317         --  $MT1 = MB * MZ$ 
318         MMMul(
319             A => MB,
320             B => MZ2,
321             C => MT1,
322             m => 4,
323             n => N,
324             p => N,
325             start_i => 3,
326             start_j => 1
327         );
328
329         --  $MT2 = a * ME$ 
330         SMMul(
331             A => ac2,
332             B => ME,
333             C => MT2,
334             m => 4,
335             n => N,
336             start_i => 3,
337             start_j => 1
338         );
339
340         --  $MA = MT1 + MT2 = MB * MZ + a * ME$ 
341         MMAdd(
342             A => MT1,
343             B => MT2,
344             C => MA,
345             m => 4,
346             n => N,
347             start_i => 3,
348             start_j => 1
349         );
350         -- Signal that  $MA_{\{H_{\{2\}}\}}$  is ready
351         Set_True(S1mah2);
352     end T2;
353
354     task T3;
355     task body T3
356     is

```

```

357         C3: Vector(1..N);
358         ac3: Integer;
359         MT1: Matrix(1..N, 1..N);
360         MT2: Matrix(1..N, 1..N);
361     begin
362         Put_Line("Task T3 started.");
363         -- Input MZ, B, ME
364         InitMatrix(MZ, N, N, 1);
365         InitVector(B, N, 1);
366         InitMatrix(ME, N, N, 1);
367         -- Signal other tasks that MZ, B, ME are ready.
368         Set_True(S1mzbme);
369         Set_True(S2mzbme);
370
371         -- Wait until C is ready
372         Suspend_Until_True(S3c);
373
374         -- Copy shared objects and compute a_3. Crit. region.
375         Suspend_Until_True(Scr);
376         Set_False(Scr);
377         C3 := C;
378         a3 := VectorMul(B, C, 5, 5);
379         -- Signal that a_3 is ready
380         Set_True(Scr);
381         Set_True(S1a3);
382
383         -- Wait until a is ready
384         Suspend_Until_True(S3a);
385
386         -- Copy a into this thread. Critical region.
387         Suspend_Until_True(Scr);
388         Set_False(Scr);
389         ac3 := a;
390         Set_True(Scr);
391
392         -- Wait until MB is ready
393         Suspend_Until_True(S3mb);
394         -- Compute MA_{H}
395         -- MT1 = MB * MZ
396         MMMul(
397             A => MB,
398             B => MZ,
399             C => MT1,
400             m => 5,
401             n => N,
402             p => N,
403             start_i => 5,

```

```

404         start_j => 1
405     );
406     -- MT2 = a * ME
407     SMMul(
408         A => ac3,
409         B => ME,
410         C => MT2,
411         m => 5,
412         n => N,
413         start_i => 5,
414         start_j => 1
415     );
416
417     -- MA = MT1 + MT2 = MB * MZ + a * ME
418     MMAAdd(
419         A => MT1,
420         B => MT2,
421         C => MA,
422         m => 5,
423         n => N,
424         start_i => 5,
425         start_j => 1
426     );
427     -- Signal that MA_{H_{3}} is ready
428     Set_True(S1mah3);
429 end T3;
430
431 begin
432     null;
433 end Run_Tasks;
434
435 begin
436     put_line("Main procedure started");
437     -- Allow access to shared resource
438     Set_True(Scr);
439     Run_Tasks;
440 end Lab01;

```
