

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

Лабораторна робота № 1.4
з дисципліни «Паралельні і розподілені обчислення»
на тему «Монітори»

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

Київ 2019

1. ЗАВДАННЯ РОБОТИ

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

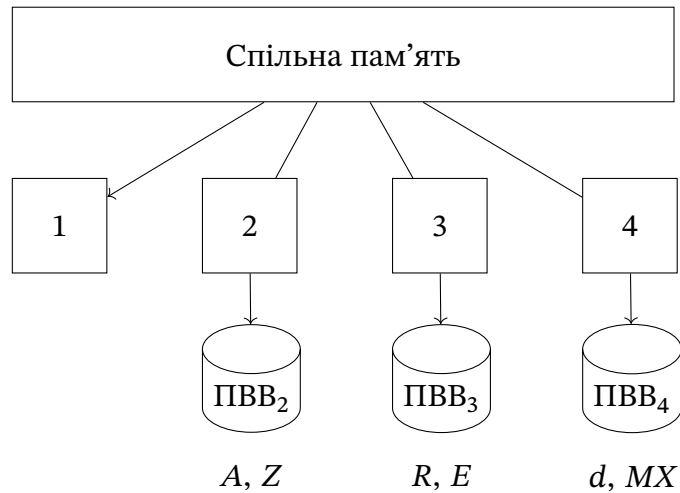


Рис. 1: Задана паралельна комп'ютерна система зі спільною пам'яттю

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

$$A = \max(Z) \cdot R + d(E \cdot MX).$$

Розробити програму на мові програмування «Python», використовуючи для взаємодії потоків (задач) монітори мови «Python» з пакету `threading` — об'єкти класу `threading.Condition`.

2. Хід роботи

2.1. Побудова паралельного алгоритму

Необхідно паралельно обчислити значення виразу $A = \max(Z) \cdot R + d(E \cdot MX)$. Для цього складаємо паралельний алгоритм:

1. Нехай існує значення загального максимуму \max_{all} . Позначимо поточний максимум як $\max_H = \max(Z_H)$.
2. У кожному потоці спробуємо змінити значення загального максимуму: $\max_{\text{all}} = \max(\max_{\text{all}}, \max_H)$.
3. Обчислюємо значення: $A_H = \max_{\text{all}} \cdot R_H + d(E \cdot MX_H)$.

Спільні ресурси: \max_{all} і d як скаляри, E — для обчислення добутку вектора на матрицю.

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
Очікувати введення Z	$W_{1,Z}$
Обчислити $max_H := \max(Z_H)$	
Обчислити $max_{all} := \max(max_H, Z)$	Критична ділянка
Дати сигнал про спробу обчислення max_{all} в задачі $T2, T3, T4$	$S_{2,M1}, S_{3,M1}, S_{4,M1}$
Чекати готовності max_{all} із задач $T2, T3, T4$	$W_{1,M2}, W_{1,M3}, W_{1,M4}$
Скопіювати $max_{all_1} := max_{all}$	Критична ділянка
Чекати готовності d, MX із задачі $T4$	$W_{1,d}, W_{1,MX}$
Скопіювати $d_1 := d$	Критична ділянка
Чекати готовності R, E із задачі $T3$	$W_{1,R}, W_{1,E}$
Скопіювати $E_1 := E$	Критична ділянка
Обчислити $A_H = max_{all_1} \cdot R_H + d_1(E_1 \cdot MX_H)$	
Дати сигнал про обчислення A_H в задачу $T2$	$S_{2,A1}$

Табл. 2: Паралельний алгоритм потоку 2

Дія	Точки синхронізації
Ввести Z	
Дати сигнал, що Z введена, в $T1, T3, T4$	$S_{1,Z}, S_{3,Z}, S_{4,Z}$
Обчислити $max_H := \max(Z_H)$	
Обчислити $max_{all} := \max(max_H, Z)$	Критична ділянка
Дати сигнал про спробу обчислення max_{all} в задачі $T1, T3, T4$	$S_{1,M2}, S_{3,M2}, S_{4,M2}$
Чекати готовності max_{all} із задач $T1, T3, T4$	$W_{2,M1}, W_{2,M3}, W_{2,M4}$
Скопіювати $max_{all_2} := max_{all}$	Критична ділянка
Чекати готовності d, MX із задачі $T4$	$W_{2,d}, W_{2,MX}$
Скопіювати $d_2 := d$	Критична ділянка
Чекати готовності R, E із задачі $T3$	$W_{2,R}, W_{2,E}$
Скопіювати $E_2 := E$	Критична ділянка
Обчислити $A_H = max_{all_2} \cdot R_H + d_2(E_2 \cdot MX_H)$	
Чекати сигналу обчислення A_H від задач $T1, T2, T3, T4$	$W_{2,A1}, W_{2,A3}, W_{2,A4}$
Вивести A	

Табл. 3: Паралельний алгоритм потоку 3

Дія	Точки синхронізації
Ввести R, E	
Дати сигнал, що R, E введені, в $T1, T2, T4$	$S_{1,R}, S_{2,R}, S_{4,R},$ $S_{1,E}, S_{2,E}, S_{4,E}$
Очікувати введення Z	$W_{2,Z}$
Обчислити $max_H := \max(Z_H)$	
Обчислити $max_{all} := \max(max_H, Z)$	Критична ділянка
Дати сигнал про спробу обчислення max_{all} в задачі $T1, T2, T4$	$S_{1,M3}, S_{2,M3}, S_{4,M3}$
Чекати готовності max_{all} із задач $T1, T2, T4$	$W_{3,M1}, W_{3,M2}, W_{3,M4}$
Скопіювати $max_{all_3} := max_{all}$	Критична ділянка
Чекати готовності d, MX із задачі $T4$	$W_{3,d}, W_{3,MX}$
Скопіювати $d_3 := d$	Критична ділянка
Обчислити $A_H = max_{all_3} \cdot R_H + d_3(E \cdot MX_H)$	
Дати сигнал про обчислення A_H в задачу $T2$	$S_{2,A3}$

Табл. 4: Паралельний алгоритм потоку 4

Дія	Точки синхронізації
Ввести d, MX	
Дати сигнал, що d, MX введені, в $T1, T2, T3$	$S_{1,d}, S_{2,d}, S_{3,d},$ $S_{1,MX}, S_{2,MX}, S_{3,MX}$
Очікувати введення Z	$W_{4,Z}$
Обчислити $max_H := \max(Z_H)$	
Обчислити $max_{all} := \max(max_H, Z)$	Критична ділянка
Дати сигнал про спробу обчислення max_{all} в задачі $T1, T2, T3$	$S_{1,M4}, S_{2,M4}, S_{3,M4}$
Чекати готовності max_{all} із задач $T1, T2, T3$	$W_{4,M1}, W_{4,M2}, W_{4,M3}$
Скопіювати $max_{all_3} := max_{all}$	Критична ділянка
Чекати готовності R, E із задачі $T3$	$W_{4,R}, W_{4,E}$
Скопіювати $E_4 := E$	Критична ділянка
Обчислити $A_H = max_{all_4} \cdot R_H + d(E_4 \cdot MX_H)$	
Дати сигнал про обчислення A_H в задачу $T2$	$S_{2,A4}$

2.3. Розробка структурної схеми взаємодії задач

Розроблюємо структурну схему взаємодії задач (рис. 2). Після розробки структурної схеми можна переходити до розробки програми.

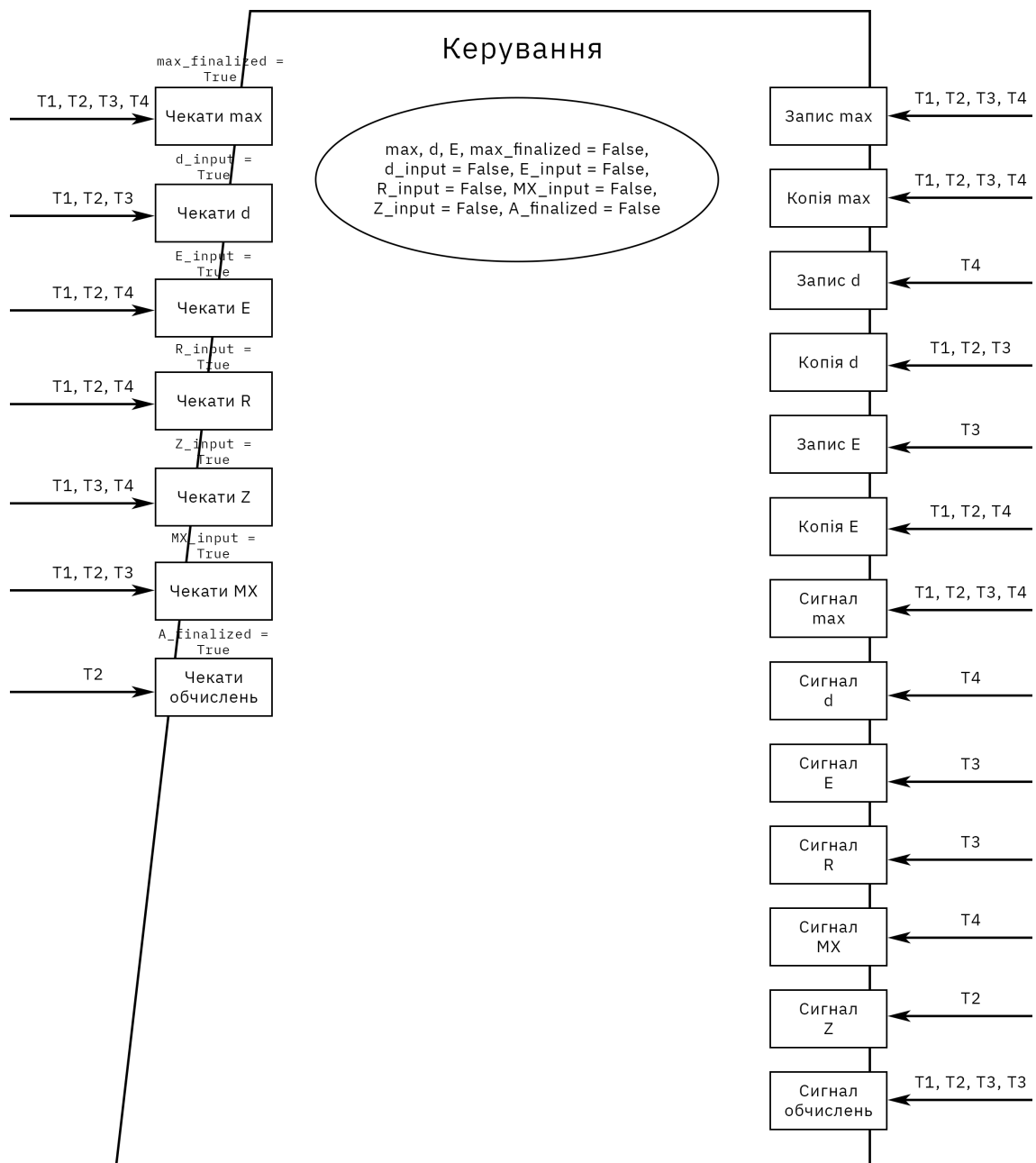
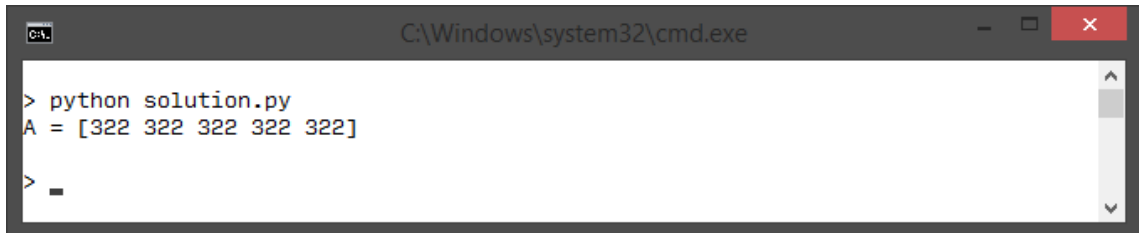


Рис. 2: Структурна схема взаємодії задач

2.4. Розробка програми

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

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe'. The command prompt shows the command '> python solution.py' followed by the output 'A = [322 322 322 322 322]'. The cursor is on a new line after the output.

```
> python solution.py
A = [322 322 322 322 322]
> _
```

Рис. 3: Результат виконання розробленої програми

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 моніторами у мові програмування «Python».

A. Програма для розв'язку поставленої задачі

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

```
1  import copy
2  import functools
3  import threading
4  import logging
5  import random
6
7  import numpy as np
8
9  logging.basicConfig(
10     level=logging.DEBUG,
11     format="%(asctime)s - %(levelname)s - %(threadName)s: %(message)s",
12 )
13
14
15  THREAD_COUNT = 4
16
17  DIMENSION = 5
18  Z_INIT_VALS = [6, 3, 2, 8, 11]
19
20  R_FILL = 2
21  E_FILL = 3
22  MX_FILL = 4
23
24
25  def synchronized(wrapped):
26     """Decorator that wraps the function to make it thread-safe: it can
27     ↪ only be
28     executed by one thread at a time.
29     """
30     lock = threading.Lock()
31     @functools.wraps(wrapped)
32     def _wrap(*args, **kwargs):
33         with lock:
34             return wrapped(*args, **kwargs)
35
36     return _wrap
37
38  def eval_A_h(A, maxall, R, d, E, MX, start, stop, dim=DIMENSION):
39     """Evaluates a part of 'A' given the passed parameters.
40
41     Args:
42         A: the object where the result will be written
```

```

43         maxall: the maxall value
44         R: the value of vector R
45         E: the value of vector E
46         MX: the value of matrix MX
47         start: starting index
48         stop: stopping index
49         dim (default: DIMENSION): element dimensions
50         """
51         R_H = R[start:stop]
52         MX_H = MX[:, start:stop]
53         A[start:stop] = maxall * R_H + d * (E.dot(MX_H))
54
55
56     class Expression(object):
57
58         def __init__(self, *args, **kwargs):
59             # Inner, protected shared resources
60             self._d = None
61             self.d_cv = threading.Condition()
62             self.d_input = False
63
64             self._E = None
65             self.E_cv = threading.Condition()
66             self.E_input = False
67
68             self._maxall = None
69             self.maxall_cv = threading.Condition()
70             self.maxall_finalized = False
71
72             self.max_counter = DIMENSION
73             self.max_counter_cv = threading.Condition()
74
75             # Signals for outer resources
76             self.Z_cv = threading.Condition()
77             self.Z_input = False
78
79             self.MX_cv = threading.Condition()
80             self.MX_input = False
81
82             self.R_cv = threading.Condition()
83             self.R_input = False
84
85             self.A_cv = threading.Condition()
86             self.A_h_counter = THREAD_COUNT
87
88         @property
89         def all_max_tried(self):

```



```

90         return self.max_counter == 0
91
92     @property
93     @synchronized
94     def maxall(self):
95         return self._maxall
96
97     @property
98     @synchronized
99     def d(self):
100         return self._d
101
102     @d.setter
103     @synchronized
104     def d(self, val):
105         with self.d_cv:
106             self._d = val
107             self.d_input = True
108             self.d_cv.notify_all()
109
110     @property
111     @synchronized
112     def E(self):
113         return self._E.copy()
114
115     @E.setter
116     @synchronized
117     def E(self, val):
118         self._E = val
119
120     @synchronized
121     def set_max(self, vec):
122         """Sets max value"""
123         with self.maxall_cv:
124             for candidate_val in vec:
125                 if self._maxall is None or candidate_val > self._maxall:
126                     self._maxall = candidate_val
127             self.max_counter = self.max_counter - len(vec)
128
129             if self.all_max_tried:
130                 self.maxall_finalized = True
131                 self.maxall_cv.notify_all()
132
133     def print_max(self):
134         with self.maxall_cv:
135             while not self.maxall_finalized:
136                 self.maxall_cv.wait()

```

```

137
138     @synchronized
139     def thread_finished_A_h(self):
140         self.A_h_counter -= 1
141         if self.A_h_counter == 0:
142             with self.A_cv:
143                 self.A_cv.notify_all()
144
145     @property
146     @synchronized
147     def A_finalized(self):
148         return self.A_h_counter == 0
149
150     def wait_maxall_finalized(self):
151         with self.maxall_cv:
152             while not self.maxall_finalized:
153                 self.maxall_cv.wait()
154             return
155
156     def signal_maxall_finalized(self):
157         with self.maxall_cv:
158             self.maxall_finalized = True
159             self.maxall_cv.notify_all()
160
161     def wait_Z_input(self):
162         with self.Z_cv:
163             while not self.Z_input:
164                 self.Z_cv.wait()
165             return
166
167     def signal_Z_input(self):
168         with self.Z_cv:
169             self.Z_input = True
170             self.Z_cv.notify_all()
171
172     def wait_d_input(self):
173         with self.d_cv:
174             while not self.d_input:
175                 self.d_cv.wait()
176             return
177
178     def signal_d_input(self):
179         with self.d_cv:
180             self.d_input = True
181             self.d_cv.notify_all()
182
183     def wait_MX_input(self):

```

```

184         with self.MX_cv:
185             while not self.MX_input:
186                 self.MX_cv.wait()
187             return
188
189     def signal_MX_input(self):
190         with self.MX_cv:
191             self.MX_input = True
192             self.MX_cv.notify_all()
193
194     def wait_R_input(self):
195         with self.R_cv:
196             while not self.R_input:
197                 self.R_cv.wait()
198             return
199
200     def signal_R_input(self):
201         with self.R_cv:
202             self.R_input = True
203             self.R_cv.notify_all()
204
205     def wait_E_input(self):
206         with self.E_cv:
207             while not self.E_input:
208                 self.E_cv.wait()
209             return
210
211     def signal_E_input(self):
212         with self.E_cv:
213             self.E_input = True
214             self.E_cv.notify_all()
215
216     def wait_A_finished(self):
217         with self.A_cv:
218             while not self.A_finalized:
219                 self.A_cv.wait()
220             return
221
222     def signal_A_finished(self):
223         with self.A_cv:
224             self.A_finalized = True
225             self.A_cv.notify_all()
226
227
228     def thread1(expression, A, Z, R, E, d, MX):
229         expression.wait_Z_input()
230         expression.set_max(Z[0:1])

```

```

231
232     expression.wait_maxall_finalized()
233
234     maxall1 = expression.maxall
235
236     expression.wait_d_input()
237     d1 = expression.d
238
239     expression.wait_E_input()
240     E1 = expression.E
241
242     eval_A_h(A, maxall1, R, d1, E1, MX, 0, 1)
243     expression.thread_finished_A_h()
244
245
246
247 def thread2(expression, A, Z, R, E, d, MX):
248     with expression.Z_cv:
249         # Initialize Z
250         for idx, val in enumerate(Z_INIT_VALS):
251             Z[idx] = val
252         # Notify subscribers
253         expression.Z_input = True
254         expression.Z_cv.notify_all()
255
256     expression.set_max(Z[1:2])
257
258     expression.wait_maxall_finalized()
259     maxall2 = expression.maxall
260
261     with expression.d_cv:
262         while not expression.d_input:
263             expression.d_cv.wait()
264             d2 = expression.d
265
266     with expression.MX_cv:
267         while not expression.MX_input:
268             expression.MX_cv.wait()
269
270     with expression.E_cv:
271         while not expression.E_input:
272             expression.MX_cv.wait()
273             E2 = expression.E
274
275     with expression.R_cv:
276         while not expression.R_input:
277             expression.R_cv.wait()

```

```

278     # Evaluate the thread's part
279     eval_A_h(A, maxall2, R, d2, E2, MX, 1, 2)
280     expression.thread_finished_A_h()
281
282     # Wait for A to finish
283     expression.wait_A_finished()
284     print("A = {}".format(A))
285
286
287 def thread3(expression, A, Z, R, E, d, MX):
288     # Initialize R
289     with expression.R_cv:
290         R.fill(R_FILL)
291         expression.R_input = True
292         expression.R_cv.notify_all()
293
294     # Initialize E
295     with expression.E_cv:
296         expression.E = np.full(DIMENSION, E_FILL)
297         E = expression.E
298         expression.E_input = True
299         expression.E_cv.notify_all()
300
301
302     with expression.Z_cv:
303         while not expression.Z_input:
304             expression.Z_cv.wait()
305
306             expression.set_max(Z[2:3])
307
308     expression.wait_maxall_finalized()
309     maxall3 = expression.maxall
310
311     expression.wait_d_input()
312     d3 = expression.d
313
314     expression.wait_MX_input()
315
316     # Evaluate the thread's part
317     eval_A_h(A, maxall3, R, d3, E, MX, 2, 3)
318     expression.thread_finished_A_h()
319
320
321 def thread4(expression, A, Z, R, E, d, MX):
322     # Input d
323     d = 5
324     expression.d = d

```

```

325
326     with expression.MX_cv:
327         # Input MX
328         MX.fill(MX_FILL)
329         # Notify subscribers
330         expression.MX_input = True
331         expression.MX_cv.notify_all()
332
333     expression.wait_Z_input()
334     expression.set_max(Z[3:5])
335
336     maxall4 = expression.maxall
337
338     expression.wait_R_input()
339     expression.wait_E_input()
340     E4 = expression.E
341
342     # Evaluate the thread's part
343     eval_A_h(A, maxall4, R, d, E4, MX, 3, 5)
344     expression.thread_finished_A_h()
345
346
347 if __name__ == "__main__":
348     expression = Expression(
349         d=0,
350         MX=0,
351         R=0,
352         E=0
353     )
354
355     A = np.ndarray(DIMENSION, np.int32)
356     R = np.ndarray(DIMENSION, np.int32)
357     E = np.ndarray(DIMENSION, np.int32)
358     Z = np.ndarray(DIMENSION, np.int32)
359     d = 0
360     MX = np.zeros((DIMENSION, DIMENSION))
361
362     t1 = threading.Thread(
363         name="t1",
364         target=thread1,
365         kwargs={
366             "expression": expression,
367             "A": A,
368             "Z": Z,
369             "R": R,
370             "E": E,
371             "d": d,

```

```

372         "MX": MX,
373     }
374 )
375
376 t2 = threading.Thread(
377     name="t2",
378     target=thread2,
379     kwargs={
380         "expression": expression,
381         "A": A,
382         "Z": Z,
383         "R": R,
384         "E": E,
385         "d": d,
386         "MX": MX,
387     }
388 )
389
390 t3 = threading.Thread(
391     name="t3",
392     target=thread3,
393     kwargs={
394         "expression": expression,
395         "A": A,
396         "Z": Z,
397         "R": R,
398         "E": E,
399         "d": d,
400         "MX": MX,
401     }
402 )
403
404 t4 = threading.Thread(
405     name="t4",
406     target=thread4,
407     kwargs={
408         "expression": expression,
409         "A": A,
410         "Z": Z,
411         "R": R,
412         "E": E,
413         "d": d,
414         "MX": MX,
415     }
416 )
417
418 t1.start()

```

```
419     t2.start()
420     t3.start()
421     t4.start()
422
423     t1.join()
424     t2.join()
425     t3.join()
426     t4.join()
```
