

Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

**Лабораторна робота №1**

“Ада. Семафори”

з дисципліни “Програмування для Паралельних Компьютерних Систем”

Виконав:

студент 3 курсу групи ІО-52

Бояршин Ігор

Номер заліковки: 5207

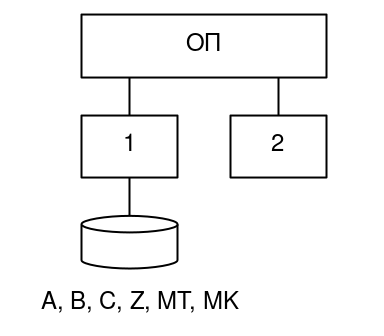
Перевірив:

Корочкін О.В.

Київ 2018 р.

**Техническое задание**

Структура паралельной компъютерной системы с общей памятью:



Выражение для подсчёта:

A = sort(B + C) + Z \* (MT \* MK)

Язык программирования:

Ада

Средства взаимодействия задач:

Семафоры

**Выполнения работы**

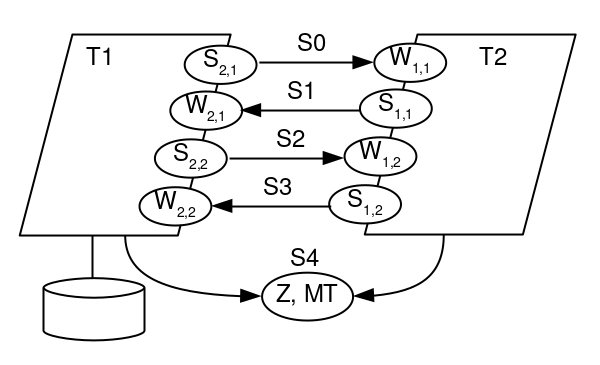
**Этап 1:** Разработка параллельного математического алгоритма

1. TH = sort(BH + CH)
2. T = sort\*(TH ; TH)
3. AH = TH + Z \* (MT \* MKH)
   * ОР: Z, MT

**Этап 2:** Разработка алгоритмов потоков

| T1 | ТС | КУ 1 | T2 | ТС | КУ 2 |
| --- | --- | --- | --- |
| Ввод B, C, Z, MT, MK |  |  |  |
| Сигнал Т2 о завершении ввода | S2,1 | Ждать завершения ввода в Т1 | W1,1 |
| Копии Z1 = Z, MT1 = MT | КУ |  |  |
| Счёт1 TH = sort(BH + CH) |  | Счёт1 TH = sort(BH + CH) |  |
| Ждать завершения счёта1 в Т2 | W2,1 | Сигнал Т1 о завершении счёта1 | S1,1 |
| Счёт2 T = sort\*(TH ; TH) |  | Копии Z2 = Z, MT2 = MT | КУ |
| Сигнал Т2 о завершении счёта2 | S2,2 | Ждать завершения счёта2 в Т1 | W1,2 |
| Счёт3 AH = TH + Z \* (MT \* MKH) |  | Счёт3 AH = TH + Z \* (MT \* MKH) |  |
| Ждать завершения счёта3 в Т2 | W2,2 | Сигнал Т1 о завершении счёта3 | S1,2 |
| Вывод A |  |  |  |

**Этап 3:** Разработка схемы взаимодействия потоков



Семафор S0 – для синхронизации по вводу.

Семафор S1 – для синхронизации по счёту1.

Семафор S2 – для синхронизации по счёту2.

Семафор S3 – для синхронизации по счёту3.

Семафор S4 – для синхронизации по копированию.

**Этап 4:** Разработка программы

**lab1.adb**

GNAT 7.3.0

Copyright 1992-2017, Free Software Foundation, Inc.

Compiling: lab1.adb

Source file time stamp: 2018-02-20 21:47:51

Compiled at: 2018-02-20 23:47:52

1. -------------------------------------------------------------------------------

2. -- Lab1: Ada. Semaphores.

3. -- Task: A = sort(B + C) + Z \* (MT \* MK)

4. -- Author: Igor Boyarshin

5. -- Date: 20.02.2018

6. -------------------------------------------------------------------------------

7. with Ada.Text\_IO, Ada.Integer\_Text\_IO, Ada.Synchronous\_Task\_Control, System.Multiprocessors, Data;

8. use Ada.Text\_IO, Ada.Integer\_Text\_IO, Ada.Synchronous\_Task\_Control, System.Multiprocessors;

9.

10. procedure Lab1

11. is

12. N: constant := 1000;

13. H: constant := N / 2;

14. Memory: constant := 500000000;

15. Output\_Threshold: constant := 5;

16.

17. package My\_Data is new Data(N);

18. use My\_Data;

19.

20. A, B, C, Z, T : Vector;

21. MK, MT : Matrix;

22. S0, S1, S2, S3, S4 : Suspension\_Object;

23.

24. procedure Tasks\_Start is

25. -- Tasks specification

26. task T1 is

27. pragma Task\_Name("Task 1");

28. pragma Storage\_Size(Memory);

29. end T1;

30.

31. task T2 is

32. pragma Task\_Name("Task 2");

33. pragma Storage\_Size(Memory);

34. end T2;

35.

36.

37. -- Tasks implementation

38. task body T1

39. is

40. Z1: Vector;

41. MT1: Matrix;

42. begin

43. Put\_Line(":> Starting Task 1...");

44.

45. -- Input

46. Fill\_Vector\_Ones(B);

47. Fill\_Vector\_Ones(C);

48. Fill\_Vector\_Ones(Z);

49. Fill\_Matrix\_Ones(MT);

50. Fill\_Matrix\_Ones(MK);

51.

52. -- Signal T2 about input end

53. Set\_True(S0);

54.

55. -- Copy Z, MT

56. Suspend\_Until\_True(S4);

57. Z1 := Copy\_Vector(Z);

58. MT1 := Copy\_Matrix(MT);

59. Set\_True(S4);

60.

61. -- Calculations 1

62. Add\_Vectors\_H(T, B, C, 1, H);

63. Sort\_Vector\_H(T, 1, H);

64.

65. -- Wait for calculations 1 end in T2

66. Suspend\_Until\_True(S1);

67.

68. -- Calculations 2

69. T := Sort\_Vector\_Merge\_Halves(T);

70.

71. -- Signal T2 about calculations 2 end

72. Set\_True(S2);

73.

74. -- Calculations 3

75. Add\_Vectors\_H(

76. A,

77. T,

78. Mul\_Vector\_Matrix\_H(

79. Z1,

80. Mul\_Matrices\_H(

81. MT1,

82. MK,

83. 1, H

84. ),

85. 1, H

86. ),

87. 1, H

88. );

89.

90. -- Wait for calculations 3 end in T2

91. Suspend\_Until\_True(S3);

92.

93. -- Output

94. if (N <= Output\_Threshold) then

95. Output\_Vector(A);

96. end if;

97.

98. Put\_Line(":> Finished Task 1");

99. end T1;

100.

101.

102. task body T2

103. is

104. Z2: Vector;

105. MT2: Matrix;

106. begin

107. Put\_Line(":> Starting Task 2...");

108.

109. -- Wait for input end in T1

110. Suspend\_Until\_True(S0);

111.

112. -- Calculations 1

113. Add\_Vectors\_H(T, B, C, H + 1, N);

114. Sort\_Vector\_H(T, H + 1, N);

115.

116. -- Signal T1 about calculations 1 end

117. Set\_True(S1);

118.

119. -- Copy Z, MT

120. Suspend\_Until\_True(S4);

121. Z2 := Copy\_Vector(Z);

122. MT2 := Copy\_Matrix(MT);

123. Set\_True(S4);

124.

125. -- Wait for calculations 2 end in T1

126. Suspend\_Until\_True(S2);

127.

128. -- Calculations 3

129. Add\_Vectors\_H(

130. A,

131. T,

132. Mul\_Vector\_Matrix\_H(

133. Z2,

134. Mul\_Matrices\_H(

135. MT2,

136. MK,

137. H + 1, N

138. ),

139. H + 1, N

140. ),

141. H + 1, N

142. );

143.

144. -- Signal T1 about calculations 3 end

145. Set\_True(S3);

146.

147. Put\_Line(":> Finished Task 2");

148. end T2;

149.

150. begin

151. null;

152. end Tasks\_Start;

153.

154.

155. begin

156. Set\_True(S4);

157. Tasks\_Start;

158. end Lab1;

158 lines: No errors

**data.adb**

GNAT 7.3.0

Copyright 1992-2017, Free Software Foundation, Inc.

Compiling: data.adb

Source file time stamp: 2018-02-20 20:51:55

Compiled at: 2018-02-20 23:47:53

1. -- Data package(implementation)

2.

3. with Ada.Text\_IO, Ada.Integer\_Text\_IO;

4. use Ada.Text\_IO, Ada.Integer\_Text\_IO;

5.

6. package body Data is

7.

8.

9. procedure Input\_Scalar(i: out Integer) is

10. begin

11. Put\_Line("Input scalar start:");

12.

13. Get(i);

14.

15. Put\_Line("Input scalar end");

16. end Input\_Scalar;

17.

18.

19. procedure Input\_Vector(V: out Vector) is

20. begin

21. Put\_Line("Input vector start:");

22.

23. for i in 1..N loop

24. Get(V(i));

25. end loop;

26.

27. Put\_Line("Input vector end");

28. end Input\_Vector;

29.

30.

31. procedure Input\_Matrix(M: out Matrix) is

32. begin

33. Put\_Line("Input matrix start:");

34.

35. for i in 1..N loop

36. for j in 1..N loop

37. Get(M(i)(j));

38. end loop;

39. end loop;

40.

41. Put\_Line("Input matrix end");

42. end Input\_Matrix;

43.

44.

45. procedure Output\_Scalar(i: in Integer) is

46. begin

47. Put\_Line("Output scalar start:");

48.

49. Put\_Line(Integer'Image(i));

50.

51. Put\_Line("Output scalar end");

52. end Output\_Scalar;

53.

54.

55. procedure Output\_Vector(V: in Vector) is

56. begin

57. Put\_Line("Output vector start:");

58.

59. for i in 1..N loop

60. Put(Integer'Image(V(i)));

61. end loop;

62. Put\_Line("");

63.

64. Put\_Line("Output vector end");

65. end Output\_Vector;

66.

67.

68. procedure Output\_Matrix(M: in Matrix) is

69. begin

70. Put\_Line("Output matrix start:");

71.

72. for i in 1..N loop

73. for j in 1..N loop

74. Put(Integer'Image(M(i)(j)));

75. end loop;

76. Put\_Line("");

77. end loop;

78.

79. Put\_Line("Output matrix end");

80. end Output\_Matrix;

81.

82.

83. procedure Fill\_Vector\_Ones(V: out Vector)

84. is

85. begin

86. for i in 1..N loop

87. V(i) := 1;

88. end loop;

89. end Fill\_Vector\_Ones;

90.

91.

92. procedure Fill\_Matrix\_Ones(M: out Matrix)

93. is

94. begin

95. for i in 1..N loop

96. for j in 1..N loop

97. M(i)(j) := 1;

98. end loop;

99. end loop;

100. end Fill\_Matrix\_Ones;

101.

102.

103. function Copy\_Vector(V: in Vector) return Vector

104. is

105. V\_Result : Vector;

106. begin

107. for i in 1..N loop

108. V\_Result(i) := V(i);

109. end loop;

110.

111. return V\_Result;

112. end Copy\_Vector;

113.

114.

115. function Copy\_Matrix(M: in Matrix) return Matrix

116. is

117. M\_Result : Matrix;

118. begin

119. for i in 1..N loop

120. for j in 1..N loop

121. M\_Result(i)(j) := M(i)(j);

122. end loop;

123. end loop;

124.

125. return M\_Result;

126. end Copy\_Matrix;

127.

128.

129. procedure Sort\_Vector\_H(V: in out Vector; start, finish: in Integer)

130. is

131. i, j: Integer;

132. tmp, pivot: Integer;

133. begin

134. i := start;

135. j := finish;

136. pivot := V((start + finish) / 2);

137.

138. while (i <= j) loop

139. while (V(i) < pivot) loop

140. i := i + 1;

141. end loop;

142. while (V(j) > pivot) loop

143. j := j - 1;

144. end loop;

145.

146. if (i <= j) then

147. tmp := V(i);

148. V(i) := V(j);

149. V(j) := tmp;

150. i := i + 1;

151. j := j - 1;

152. end if;

153. end loop;

154.

155. if (start < j) then

156. Sort\_Vector\_H(V, start, j);

157. end if;

158. if (i < finish) then

159. Sort\_Vector\_H(V, i, finish);

160. end if;

161. end Sort\_Vector\_H;

162.

163.

164. function Sort\_Vector\_Merge\_Halves(V: in Vector) return Vector

165. is

166. i, j, index: Integer;

167. V\_Result: Vector;

168. begin

169. i := 1;

170. j := N / 2 + 1;

171. index := 1;

172. while (index <= N) loop

173. if (i >= N / 2 + 1) then

174. V\_Result(index) := V(j);

175. j := j + 1;

176. elsif (j >= N + 1) then

177. V\_Result(index) := V(i);

178. i := i + 1;

179. elsif (V(i) < V(j)) then

180. V\_Result(index) := V(i);

181. i := i + 1;

182. else

183. V\_Result(index) := V(j);

184. j := j + 1;

185. end if;

186.

187. index := index + 1;

188. end loop;

189.

190. return V\_Result;

191. end Sort\_Vector\_Merge\_Halves;

192.

193.

194. function Mul\_Vector\_Matrix\_H(

195. V: in Vector; M: in Matrix; start, finish: in Integer)

196. return Vector

197. is

198. V\_Result : Vector;

199. begin

200. for i in start..finish loop

201. V\_Result(i) := 0;

202. for j in 1..N loop

203. V\_Result(i) := V\_Result(i) + V(j) \* M(j)(i);

204. end loop;

205. end loop;

206.

207. return V\_Result;

208. end Mul\_Vector\_Matrix\_H;

209.

210.

211. function Mul\_Matrices\_H(

212. M1, M2: in Matrix; start, finish: in Integer)

213. return Matrix

214. is

215. M\_Result : Matrix;

216. begin

217. for row in 1..N loop

218. for column in start..finish loop

219. M\_Result(row)(column) := 0;

220. for i in 1..N loop

221. M\_Result(row)(column) :=

222. M\_Result(row)(column) + M1(row)(i) \* M2(i)(column);

223. end loop;

224. end loop;

225. end loop;

226.

227. return M\_Result;

228. end Mul\_Matrices\_H;

229.

230.

231. procedure Add\_Vectors\_H(

232. V\_Result: out Vector; V1, V2: in Vector; start, finish: in Integer)

233. is

234. begin

235. for i in start..finish loop

236. V\_Result(i) := V1(i) + V2(i);

237. end loop;

238. end Add\_Vectors\_H;

239.

240.

241. end Data;

**data.ads**

Compiling: data.ads

Source file time stamp: 2018-02-20 20:52:03

Compiled at: 2018-02-20 23:47:53

1. -- Data package(specification)

2.

3. generic

4. N: Integer;

5. package Data is

6.

7. type Vector is private;

8. type Matrix is private;

9.

10.

11. procedure Input\_Scalar (i: out Integer);

12. procedure Input\_Vector (V: out Vector);

13. procedure Input\_Matrix (M: out Matrix);

14. procedure Output\_Scalar(i: in Integer);

15. procedure Output\_Vector(V: in Vector);

16. procedure Output\_Matrix(M: in Matrix);

17.

18.

19. procedure Fill\_Vector\_Ones(V: out Vector);

20. procedure Fill\_Matrix\_Ones(M: out Matrix);

21.

22. function Copy\_Vector(V: in Vector) return Vector;

23. function Copy\_Matrix(M: in Matrix) return Matrix;

24.

25.

26. procedure Sort\_Vector\_H(V: in out Vector; start, finish: in Integer);

27. function Sort\_Vector\_Merge\_Halves(V: in Vector) return Vector;

28. function Mul\_Vector\_Matrix\_H(

29. V: in Vector; M: in Matrix; start, finish: in Integer)

30. return Vector;

31. function Mul\_Matrices\_H(

32. M1, M2: in Matrix; start, finish: in Integer)

33. return Matrix;

34. procedure Add\_Vectors\_H(

35. V\_Result: out Vector; V1, V2: in Vector; start, finish: in Integer);

36.

37.

38. private

39. type Vector is array (1..N) of Integer;

40. type Matrix is array (1..N) of Vector;

41.

42. end Data;

241 lines: No errors