

Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

**Лабораторна робота №5**

“Ада. Защищённые модули”

з дисципліни “Програмування для Паралельних Компьютерних Систем”

Виконав:

студент 3 курсу групи ІО-52

Бояршин Ігор

Номер заліковки: 5207

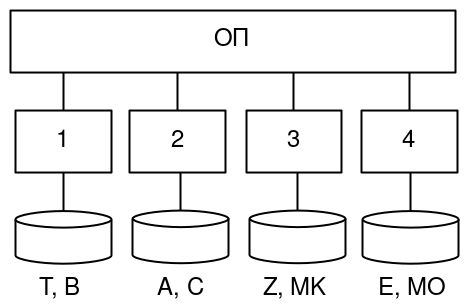
Перевірив:

Корочкін О.В.

Київ 2018 р.

**Техническое задание**

Структура паралельной компъютерной системы с общей памятью:



Выражение для подсчёта:

A = (B\*C)\*Z + min(E) \* T \* (MO\*MK)

Язык программирования:

Ада

Средства взаимодействия задач:

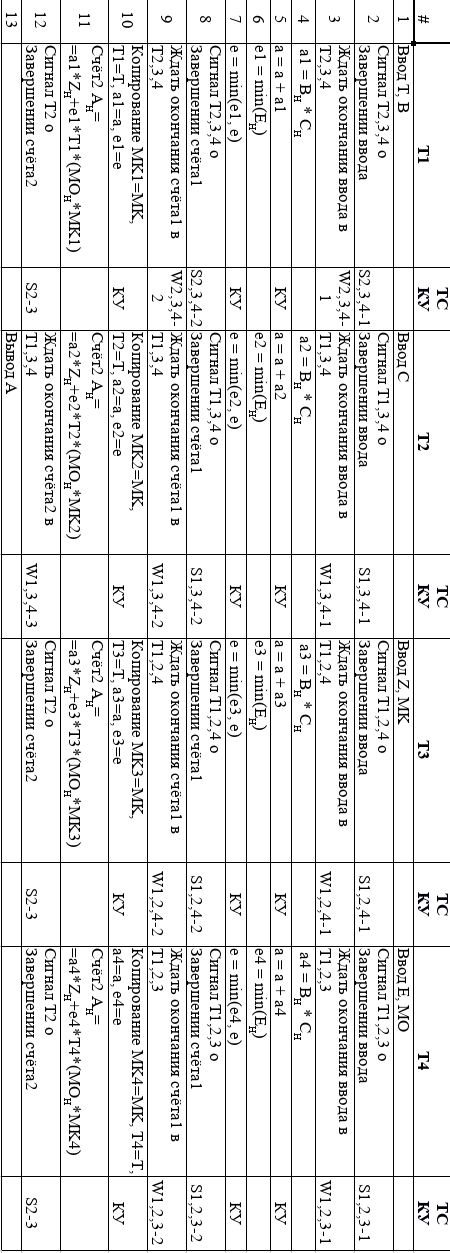
Защищённые модули

**Выполнение работы**

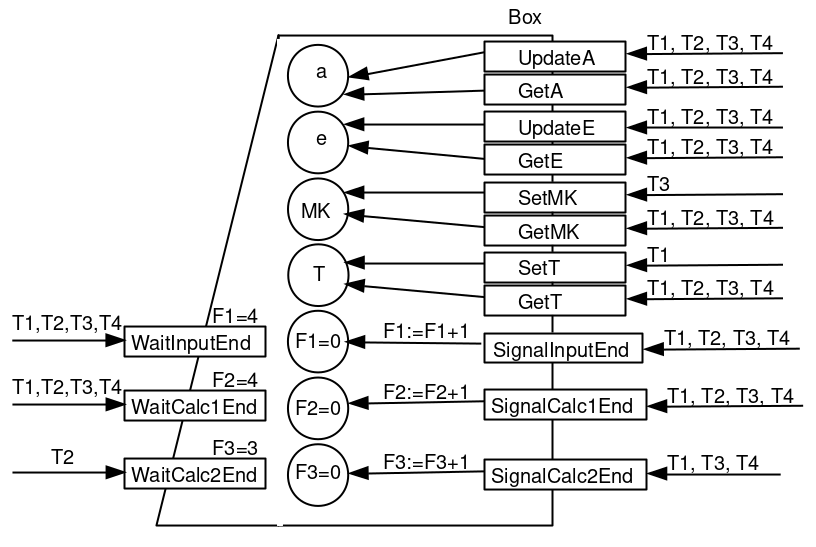
**Этап 1:** Разработка параллельного математического алгоритма

1. ai = BH \* CH , i = (1..P), P – количество потоков, P = 4
2. a = a + ai , i = (1..P)
   * ОР: а
3. ei = min(EH) , i = (1..P)
4. e = min(e; ei) , i = (1..P)
   * ОР: е
5. AH = b \* ZH + e \* T \* (MOH \* MK)
   * ОР: T, MK, b, e

**Этап 2:** Разработка алгоритмов потоков



**Этап 3:** Разработка схемы взаимодействия потоков



**Этап 4:** Разработка программы

GNAT 7.3.1 20180312

Copyright 1992-2017, Free Software Foundation, Inc.

Compiling: lab5.adb

Source file time stamp: 2018-04-15 16:30:42

Compiled at: 2018-04-15 19:30:47

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

2. -- Lab5: Ada. Protected Modeules

3. -- Task: (B\*C)\*Z + min(E) \* T \* (MO\*MK)

4. -- Author: Igor Boyarshin

5. -- Date: 14.04.2018

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 Lab5

11. is

12. N: constant := 3000;

13. P: constant := 4;

14. H: constant := N / P;

15. Memory: constant := 5000000;

16. Output\_Threshold: constant := 8;

17.

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

19. use My\_Data;

20.

21. A: access Vector := new Vector;

22. B: access Vector := new Vector;

23. C: access Vector := new Vector;

24. Z: access Vector := new Vector;

25. E: access Vector := new Vector;

26. -- T: access Vector := new Vector;

27. MO: access Matrix := new Matrix;

28. -- MK: access Matrix := new Matrix;

29.

30. -------------------------------------------------------------------------------

31. -- Protected Module Specification

32. protected Box is

33. procedure UpdateA(x: in Integer);

34. function GetA return Integer;

35. procedure UpdateE(x: in Integer);

36. function GetE return Integer;

37. procedure SetMK(M: access Matrix);

38. function GetMK return access Matrix;

39. procedure SetT(V: access Vector);

40. function GetT return access Vector;

41. procedure SignalInputEnd;

42. procedure SignalCalc1End;

43. procedure SignalCalc2End;

44. entry WaitInputEnd;

45. entry WaitCalc1End;

46. entry WaitCalc2End;

47.

48. private

49. a: Integer := 0;

50. e: Integer := 32000;

51. MK: access Matrix := new Matrix;

52. T: access Vector := new Vector;

53. F1: Integer := 0;

54. F2: Integer := 0;

55. F3: Integer := 0;

56. end Box;

57.

58. -- Protected Module Implementation

59. protected body Box is

60. procedure UpdateA(x: in Integer) is

61. begin

62. a := a + x;

63. end UpdateA;

64.

65. function GetA return Integer is

66. begin

67. return a;

68. end GetA;

69.

70. procedure UpdateE(x: in Integer) is

71. begin

72. if (x < e) then

73. e := x;

74. end if;

75. end UpdateE;

76.

77. function GetE return Integer is

78. begin

79. return e;

80. end GetE;

81.

82. procedure SetMK(M: access Matrix) is

83. begin

84. MK := M;

85. end SetMK;

86.

87. function GetMK return access Matrix is

88. begin

89. return Copy\_Matrix(MK);

90. end;

91.

92. procedure SetT(V: access Vector) is

93. begin

94. T := V;

95. end SetT;

96.

97. function GetT return access Vector is

98. begin

99. return Copy\_Vector(T);

100. end GetT;

101.

102. procedure SignalInputEnd is

103. begin

104. F1 := F1 + 1;

105. end SignalInputEnd;

106.

107. procedure SignalCalc1End is

108. begin

109. F2 := F2 + 1;

110. end SignalCalc1End;

111.

112. procedure SignalCalc2End is

113. begin

114. F3 := F3 + 1;

115. end SignalCalc2End;

116.

117. entry WaitInputEnd when F1 = 4 is

118. begin

119. null;

120. end WaitInputEnd;

121.

122. entry WaitCalc1End when F2 = 4 is

123. begin

124. null;

125. end WaitCalc1End;

126.

127. entry WaitCalc2End when F3 = 3 is

128. begin

129. null;

130. end WaitCalc2End;

131. end Box;

132. -------------------------------------------------------------------------------

133. procedure Tasks\_Start is

134. -- Tasks specification

135. task T1 is

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

137. pragma Storage\_Size(Memory);

138. end T1;

139. task T2 is

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

141. pragma Storage\_Size(Memory);

142. end T2;

143. task T3 is

144. pragma Task\_Name("Task 3");

145. pragma Storage\_Size(Memory);

146. end T3;

147. task T4 is

148. pragma Task\_Name("Task 4");

149. pragma Storage\_Size(Memory);

150. end T4;

151. -------------------------------------------------------------------------------

152. -- Tasks implementation

153. task body T1

154. is

155. low, high: Integer;

156. temp, prod: Integer;

157. ai, ei: Integer;

158. Ti: access Vector := new Vector;

159. MKi: access Matrix;

160. begin

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

162. low := 1;

163. high := H;

164.

165. -- Input

166. Fill\_Vector\_Ones(Ti);

167. Box.SetT(Ti);

168. Fill\_Vector\_Ones(B);

169.

170. -- Synchronize on input

171. Box.SignalInputEnd;

172. Box.WaitInputEnd;

173.

174. -- Calculations1 A

175. ai := 0;

176. for i in low..high loop

177. ai := ai + B(i) \* C(i);

178. end loop;

179.

180. Box.UpdateA(ai);

181.

182. -- Calculations1 E

183. ei := E(low);

184. for i in low..high loop

185. if (E(i) < ei) then

186. ei := E(i);

187. end if;

188. end loop;

189.

190. Box.UpdateE(ei);

191.

192. -- Synchronize on Calculations1

193. Box.SignalCalc1End;

194. Box.WaitCalc1End;

195.

196. -- Copies

197. ai := Box.GetA;

198. ei := Box.GetE;

199. MKi := Box.GetMK;

200. Ti := Box.GetT;

201.

202. -- Calculations 2

203. for h in low..high loop

204. temp := 0;

205. for i in 1..N loop

206. prod := 0;

207. for j in 1..N loop

208. prod := prod + MO(h)(j) \* MKi(j)(i);

209. end loop;

210.

211. temp := temp + Ti(i) \* prod;

212. end loop;

213.

214. A(h) := ai \* Z(h) + ei \* temp;

215. end loop;

216.

217. -- Synchronize on Calculations2

218. Box.SignalCalc2End;

219.

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

221. end T1;

222. -------------------------------------------------------------------------------

223. task body T2

224. is

225. low, high: Integer;

226. temp, prod: Integer;

227. ai, ei: Integer;

228. Ti: access Vector;

229. MKi: access Matrix;

230. begin

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

232. low := H + 1;

233. high := 2\*H;

234.

235. -- Input

236. Fill\_Vector\_Ones(C);

237. for i in 1..N loop

238. A(i) := 0;

239. end loop;

240.

241. -- Synchronize on input

242. Box.SignalInputEnd;

243. Box.WaitInputEnd;

244.

245. -- Calculations1 A

246. ai := 0;

247. for i in low..high loop

248. ai := ai + B(i) \* C(i);

249. end loop;

250.

251. Box.UpdateA(ai);

252.

253. -- Calculations1 E

254. ei := E(low);

255. for i in low..high loop

256. if (E(i) < ei) then

257. ei := E(i);

258. end if;

259. end loop;

260.

261. Box.UpdateE(ei);

262.

263. -- Synchronize on Calculations1

264. Box.SignalCalc1End;

265. Box.WaitCalc1End;

266.

267. -- Copies

268. ai := Box.GetA;

269. ei := Box.GetE;

270. MKi := Box.GetMK;

271. Ti := Box.GetT;

272.

273. -- Calculations 2

274. for h in low..high loop

275. temp := 0;

276. for i in 1..N loop

277. prod := 0;

278. for j in 1..N loop

279. prod := prod + MO(h)(j) \* MKi(j)(i);

280. end loop;

281.

282. temp := temp + Ti(i) \* prod;

283. end loop;

284.

285. A(h) := ai \* Z(h) + ei \* temp;

286. end loop;

287.

288. -- Synchronize on Calculations2

289. Box.WaitCalc2End;

290.

291. -- Output

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

293. Output\_Vector(A);

294. end if;

295.

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

297. end T2;

298. -------------------------------------------------------------------------------

299. task body T3

300. is

301. low, high: Integer;

302. temp, prod: Integer;

303. ai, ei: Integer;

304. Ti: access Vector;

305. MKi: access Matrix := new Matrix;

306. begin

307. Put\_Line(":> Starting Task 3...");

308. low := 2\*H + 1;

309. high := 3\*H;

310.

311. -- Input

312. Fill\_Vector\_Ones(Z);

313. Fill\_Matrix\_Ones(MKi);

314. Box.SetMK(MKi);

315.

316. -- Synchronize on input

317. Box.SignalInputEnd;

318. Box.WaitInputEnd;

319.

320. -- Calculations1 A

321. ai := 0;

322. for i in low..high loop

323. ai := ai + B(i) \* C(i);

324. end loop;

325.

326. Box.UpdateA(ai);

327.

328. -- Calculations1 E

329. ei := E(low);

330. for i in low..high loop

331. if (E(i) < ei) then

332. ei := E(i);

333. end if;

334. end loop;

335.

336. Box.UpdateE(ei);

337.

338. -- Synchronize on Calculations1

339. Box.SignalCalc1End;

340. Box.WaitCalc1End;

341.

342. -- Copies

343. ai := Box.GetA;

344. ei := Box.GetE;

345. MKi := Box.GetMK;

346. Ti := Box.GetT;

347.

348. -- Calculations 2

349. for h in low..high loop

350. temp := 0;

351. for i in 1..N loop

352. prod := 0;

353. for j in 1..N loop

354. prod := prod + MO(h)(j) \* MKi(j)(i);

355. end loop;

356.

357. temp := temp + Ti(i) \* prod;

358. end loop;

359.

360. A(h) := ai \* Z(h) + ei \* temp;

361. end loop;

362.

363. -- Synchronize on Calculations2

364. Box.SignalCalc2End;

365.

366. Put\_Line(":> Finished Task 3");

367. end T3;

368. -------------------------------------------------------------------------------

369. task body T4

370. is

371. low, high: Integer;

372. temp, prod: Integer;

373. ai, ei: Integer;

374. Ti: access Vector;

375. MKi: access Matrix;

376. begin

377. Put\_Line(":> Starting Task 4...");

378. low := 3\*H + 1;

379. high := 4\*H;

380.

381. -- Input

382. Fill\_Vector\_Ones(E);

383. Fill\_Matrix\_Ones(MO);

384.

385. -- Synchronize on input

386. Box.SignalInputEnd;

387. Box.WaitInputEnd;

388.

389. -- Calculations1 A

390. ai := 0;

391. for i in low..high loop

392. ai := ai + B(i) \* C(i);

393. end loop;

394.

395. Box.UpdateA(ai);

396.

397. -- Calculations1 E

398. ei := E(low);

399. for i in low..high loop

400. if (E(i) < ei) then

401. ei := E(i);

402. end if;

403. end loop;

404.

405. Box.UpdateE(ei);

406.

407. -- Synchronize on Calculations1

408. Box.SignalCalc1End;

409. Box.WaitCalc1End;

410.

411. -- Copies

412. ai := Box.GetA;

413. ei := Box.GetE;

414. MKi := Box.GetMK;

415. Ti := Box.GetT;

416.

417. -- Calculations 2

418. for h in low..high loop

419. temp := 0;

420. for i in 1..N loop

421. prod := 0;

422. for j in 1..N loop

423. prod := prod + MO(h)(j) \* MKi(j)(i);

424. end loop;

425.

426. temp := temp + Ti(i) \* prod;

427. end loop;

428.

429. A(h) := ai \* Z(h) + ei \* temp;

430. end loop;

431.

432. -- Synchronize on Calculations2

433. Box.SignalCalc2End;

434.

435. Put\_Line(":> Finished Task 4");

436. end T4;

437. -------------------------------------------------------------------------------

438.

439. begin

440. null;

441. end Tasks\_Start;

442. begin

443. Tasks\_Start;

444. end Lab5;

444 lines: No errors