ДОДАТКИ

1. Структурна схема ПКС СП
2. Схема алгоритму головної програми зі вказанням паралельних ділянок для ПРГ1
3. Схема алгоритму процесів для програми ПРГ1
4. Лістинг програми ПРГ1

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

2. --

3. -- Parallel and Distributed Bomputing

4. -- Laboratory work #5. Ada. Protected unit

5. --

6. -- File: pro2\_lab5.adb

7. -- Task: A = (B + C \* MO)(MX \* MZ)

8. --

9. -- Author: Shiyka Vladislav, group IO-01

10. -- Date: 17.04.2013

11. --

12. ---------------------------------------------------------------

13.

14. with Ada.Text\_IO;

15. use Ada.Text\_IO;

16.

17. generic

18.

19. N : in Natural; -- dimension of Vector and Matrix(N \* N)

20.

21. package Data is

22.

23. type Vector is array (1..N) of integer;

24. type Matrix is array (1..n) of Vector;

25.

26. procedure Input (Value : in Integer;

27. V : out Vector);

28.

29. procedure Input (Value : in Integer;

30. MA : out Matrix);

31.

32. procedure Output (V : in Vector);

33.

34. procedure Output (MA : in Matrix);

35.

36. function mul

37. (Left : Matrix;

38. Right : Matrix;

39. L : Natural;

40. R : Natural) return Matrix;

41.

42. function Mul

43. (Left : Vector;

44. Right : Matrix;

45. L : Natural;

46. R : Natural) return Vector;

47.

48. function Add

49. (Left : Vector;

50. Right : Vector;

51. L : Natural;

52. R : Natural) return Vector;

53.

54.

55. end Data;

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

2. --

3. -- Parallel and Distributed Bomputing

4. -- Laboratory work #5. Ada. Protected unit

5. --

6. -- File: pro2\_lab5.adb

7. -- Task: A = (B + C \* MO)(MX \* MZ)

8. --

9. -- Author: Shiyka Vladislav, group IO-01

10. -- Date: 17.04.2013

11. --

12. --------------------------------------------------------------

13.

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

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

16.

17. package body Data is

18.

19. procedure Input (Value : Integer; V : out Vector) is

20. begin

21. for I in 1..N loop

22. V(I) := Value;

23. end loop;

24. end Input;

25.

26.

27. procedure Input (Value : Integer; MA : out Matrix) is

28. begin

29. for I in 1..N loop

30. for J in 1..N loop

31. MA(I)(J) := Value;

32. end loop;

33. end loop;

34. end Input;

35.

36.

37. procedure Output (V : in Vector) is

38. begin

39. New\_Line;

40. for I in 1..N loop

41. Put(Item => V(I), Width => 6);

42. put("|");

43. end loop;

44. New\_Line;

45. end Output;

46.

47.

48. procedure Output (MA : in Matrix) is

49. begin

50. New\_Line;

51. for I in 1..N loop

52. for J in 1..N loop

53. Put(Item => MA(i)(j), Width => 6);

54. put("||");

55. end loop;

56. New\_line;

57. end loop;

58. New\_Line;

59. end Output;

60.

61.

62. function mul

63. (Left : Matrix;

64. Right : Matrix;

65. L : Natural;

66. R : Natural) return Matrix

67. is

68. MR : Matrix;

69. begin

70. for i in 1..N loop

71. for J in L..R loop

72. MR(I)(J) := 0;

73. for K in 1..N loop

74. MR(I)(J):=MR(I)(J)+Left(I)(K)\*Right(K)(J);

75. end loop;

76. end loop;

77. end loop;

78. return MR;

79. end Mul;

80.

81.

82. function Mul

83. (Left : Vector;

84. Right : Matrix;

85. L : Natural;

86. R : Natural) return Vector

87. is

88. V : Vector;

89. begin

90. for J in L..R loop

91. V(j) := 0;

92. begin

93. for K in 1..N loop

94. V(J) := V(J) + Left(K) \* Right(K)(J);

95. end loop;

96. end;

97. end loop;

98. return V;

99. end Mul;

100.

101.

102. function Add

103. (Left : Vector;

104. Right : Vector;

105. L : Natural;

106. R : Natural) return Vector

107. is

108. V : Vector;

109. begin

110. for J in L..R loop

111. V(J) := Left (J) + Right (J);

112. end loop;

113. return V;

114. end Add;

115.

116.

117. end Data;

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

2. --

3. -- Parallel and Distributed Bomputing

4. -- Laboratory work #5. Ada. Protected unit

5. --

6. -- File: pro2\_lab5.adb

7. -- Task: A = (B + C \* MO)(MX \* MZ)

8. --

9. -- Author: Shiyka Vladislav, group IO-01

10. -- Date: 17.04.2013

11. --

12. --------------------------------------------------------------

13. with Ada.Text\_IO, Ada.Integer\_text\_iO, Ada.Synchronous\_Task\_Control, Ada.Command\_Line, Data;

14. use Ada.Text\_IO, Ada.Integer\_text\_iO, Ada.Synchronous\_Task\_Control, Ada.Command\_Line;

15.

16. procedure Pro2CW is

17.

18. Value : Integer := 1;

19. N : Integer := Integer'Value(Argument(1));

20.

21. package DataN is new Data(N);

22. use DataN;

23.

24. P : Natural := 4;

25. H : Natural := N/P;

26.

27. A, B : Vector;

28. MO, MZ : Matrix;

29.

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

31. -- ОПИСОВА ЧАСТИНА ЗАХИЩЕНИХ МОДУЛІВ

32. --------------------------------------------------------------

33.

34. -- ЗАХИЩЕНИЙ МОДУЛЬ InputSync

35. protected InputSync is

36. entry WaitForInput;

37. procedure InputSignal;

38. private

39. F : Natural := 0;

40. end InputSync;

41.

42. -- ЗАХИЩЕНИЙ МОДУЛЬ Sync1

43. protected Sync1 is

44. entry WaitForSync1;

45. procedure SignalSync1;

46. private

47. F : Natural := 0;

48. end Sync1;

49.

50. -- ЗАХИЩЕНИЙ МОДУЛЬ Sync1

51. protected Sync2 is

52. entry WaitForSync2;

53. procedure SignalSync2;

54. private

55. F : Natural := 0;

56. end Sync2;

57.

58. -- ЗАХИЩЕНИЙ МОДУЛЬ OutputSync

59. protected OutputSync is

60. entry WaitForOutput;

61. procedure OutputSignal;

62. private

63. F : Natural := 0;

64. end OutputSync;

65.

66. -- ЗАХИЩЕНИЙ МОДУЛЬ ResC

67. protected ResC is

68. procedure setC (V : in Vector);

69. function getC return Vector;

70. private

71. C : Vector;

72. end ResC;

73.

74. -- ЗАХИЩЕНИЙ МОДУЛЬ ResD

75. protected ResD is

76. procedure setD (V : in Vector; L, R : in Natural);

77. function getD return Vector;

78. private

79. D : Vector;

80. end ResD;

81.

82. -- ЗАХИЩЕНИЙ МОДУЛЬ ResMX

83. protected ResMX is

84. procedure setMX (M : in Matrix);

85. function getMX return Matrix;

86. private

87. MX : Matrix;

88. end ResMX;

89.

90. --------------------------------------------------------------

91. -- ТІЛА ЗАХИЩЕНИХ МОДУЛІВ

92. --------------------------------------------------------------

93.

94. -- ЗАХИЩЕНИЙ МОДУЛЬ InputSync

95. protected body InputSync is

96. entry WaitForInput when F >= 3 is

97. begin

98. null;

99. end WaitForInput;

100.

101. procedure InputSignal is

102. begin

103. F := F + 1;

104. end InputSignal;

105. end InputSync;

106.

107. -- ЗАХИЩЕНИЙ МОДУЛЬ Sync1

108. protected body Sync1 is

109. entry WaitForSync1 when F >= 4 is

110. begin

111. null;

112. end WaitForSync1;

113.

114. procedure SignalSync1 is

115. begin

116. F := F + 1;

117. end SignalSync1;

118. end Sync1;

119.

120. protected body Sync2 is

121. entry WaitForSync2 when F >= 4 is

122. begin

123. null;

124. end WaitForSync2;

125.

126. procedure SignalSync2 is

127. begin

128. F := F + 1;

129. end SignalSync2;

130. end Sync2;

131.

132. -- ЗАХИЩЕНИЙ МОДУЛЬ OutputSync

133. protected body OutputSync is

134. entry WaitForOutput when F >= 3 is

135. begin

136. if N <= 9 then

137. Output (A);

138. end if;

139. end WaitForOutput;

140.

141. procedure OutputSignal is

142. begin

143. F := F + 1;

144. end OutputSignal;

145. end OutputSync;

146.

147. -- ЗАХИЩЕНИЙ МОДУЛЬ ResB

148. protected body ResC is

149. procedure setC (V : in Vector) is

150. begin

151. for I in 1..N loop

152. C(I) := V(I);

153. end loop;

154. end SetC;

155.

156. function getC return Vector is

157. begin

158. return C;

159. end GetC;

160. end ResC;

161.

162. -- ЗАХИЩЕНИЙ МОДУЛЬ ResD

163. protected body ResD is

164. procedure setD (V : in Vector; L, R : in Natural) is

165. begin

166. for I in L..R loop

167. D(I) := V(I);

168. end loop;

169. end SetD;

170.

171. function getD return Vector is

172. begin

173. return D;

174. end GetD;

175. end ResD;

176.

177. -- ЗАХИЩЕНИЙ МОДУЛЬ ResMX

178. protected body ResMX is

179. procedure setMX (M : in Matrix) is

180. begin

181. for I in 1..N loop

182. for J in 1..N loop

183. MX(I)(J) := M(I)(J);

184. end loop;

185. end loop;

186. end SetMX;

187.

188. function getMX return Matrix is

189. begin

190. return MX;

191. end GetMX;

192. end ResMX;

193.

194. procedure Start is

195. -----------------------------------------------------------

196. -- ЗАДАЧА 1

197. -----------------------------------------------------------

198. task T1;

199. task body T1 is

200. C1, D1, X1, TmpA : Vector;

201. MX1, MM1 : Matrix;

202. begin

203. Put\_Line ("T1 started");

204.

205. InputSync.WaitForInput;

206. --put("TASK 1:");

207. Input(0, X1);

208. Input(0, MM1);

209. --Output(X1);

210.

211. C1 := ResC.getC;

212.

213. X1 := Mul(C1, MO, 1 ,H);

214. MX1 := ResMX.GetMX;

215. MM1 := mul(MX1, MZ, 1, H);

216.

217. Sync1.SignalSync1;

218. Sync1.WaitForSync1;

219.

220. ResD.setD(add(B, X1, 1, H), 1, H);

221.

222. Sync2.SignalSync2;

223. Sync2.WaitForSync2;

224.

225. D1 := ResD.GetD;

226. TmpA := mul(D1, MM1, 1, H);

227.

228. for I in 1..H loop

229. A(I) := TmpA(I);

230. end loop;

231.

232. OutputSync.OutputSignal;

233. Put\_Line ("T1 finished");

234. end T1;

235.

236. -----------------------------------------------------------

237. -- ЗАДАЧА 2

238. -----------------------------------------------------------

239. task T2;

240. task body T2 is

241. C2, D2, X2, TmpA : Vector;

242. MX2, MM2 : Matrix;

243. begin

244. Put\_Line ("T2 started");

245.

246. Input (Value, B);

247. Input (Value, C2);

248. ResC.SetC(C2);

249.

250. InputSync.InputSignal;

251. InputSync.WaitForInput;

252. --put("TASK 2:");

253. Input(0, X2);

254. Input(0, MM2);

255. --Output(X2);

256.

257. C2 := ResC.getC;

258.

259. X2 := Mul(C2, MO, H+1, 2\*H);

260. MX2 := ResMX.GetMX;

261. MM2 := mul(MX2, MZ, H+1, 2\*H);

262.

263. Sync1.SignalSync1;

264. Sync1.WaitForSync1;

265.

266. ResD.setD(add(B, X2, H+1, 2\*H), H+1, 2\*H);

267.

268. Sync2.SignalSync2;

269. Sync2.WaitForSync2;

270.

271. D2 := ResD.getD;

272. TmpA := mul(D2, MM2, H+1, 2\*H);

273.

274. for I in H+1..2\*H loop

275. A(I) := TmpA(I);

276. end loop;

277.

278. OutputSync.WaitForOutput;

279.

280. Put\_Line ("T2 finished");

281. end T2;

282.

283. -----------------------------------------------------------

284. -- ЗАДАЧА 3

285. -----------------------------------------------------------

286. task T3;

287. task body T3 is

288. C3, D3, X3, TmpA : Vector;

289. MX3, MM3 : Matrix;

290. begin

291. Put\_Line ("T3 started");

292.

293. Input (Value, MO);

294. Input (Value, MX3);

295. ResMX.setMX(MX3);

296.

297. InputSync.InputSignal;

298. InputSync.WaitForInput;

299. --put("TASK 3:");

300. Input(0, X3);

301. Input(0, MM3);

302. --Output(X3);

303.

304. C3 := ResC.getC;

305.

306. X3 := Mul(C3, MO, H+1, 2\*H);

307. MX3 := ResMX.GetMX;

308. MM3 := mul(MX3, MZ, 2\*H+1, 3\*H);

309.

310. Sync1.SignalSync1;

311. Sync1.WaitForSync1;

312.

313. ResD.setD(add(B, X3, 2\*H+1, 3\*H), 2\*H+1, 3\*H);

314.

315. Sync2.SignalSync2;

316. Sync2.WaitForSync2;

317.

318. D3 := ResD.GetD;

319. TmpA := mul(D3, MM3, 2\*H+1, 3\*H);

320.

321. for I in 2\*H+1..3\*H loop

322. A(I) := TmpA(I);

323. end loop;

324.

325. OutputSync.OutputSignal;

326.

327. Put\_Line ("T3 finished");

328. end T3;

329.

330. -----------------------------------------------------------

331. -- ЗАДАЧА 4

332. -----------------------------------------------------------

333. task T4;

334. task body T4 is

335. C4, D4, X4, TmpA : Vector;

336. MX4,MM4 : Matrix;

337. begin

338. Put\_Line ("T4 started");

339.

340. Input (Value, MZ);

341.

342. InputSync.InputSignal;

343. InputSync.WaitForInput;

344. --put("TASK 4:");

345. Input(0, X4);

346. Input(0, MM4);

347. --Output(X4);

348.

349. C4 := ResC.getC;

350.

351. X4 := Mul(C4, MO, 3\*H+1, N);

352. MX4 := ResMX.GetMX;

353. MM4 := mul(MX4, MZ, 3\*H+1, N);

354.

355. Sync1.SignalSync1;

356. Sync1.WaitForSync1;

357.

358. ResD.setD(add(B, X4, 3\*H+1, N), 3\*H+1, N);

359.

360. Sync2.SignalSync2;

361. Sync2.WaitForSync2;

362.

363. D4 := ResD.GetD;

364. TmpA := mul(D4, MM4, 3\*H+1, N);

365.

366. for I in 3\*H+1..N loop

367. A(I) := TmpA(I);

368. end loop;

369.

370. OutputSync.OutputSignal;

371.

372. Put\_Line ("T4 finished");

373. end T4;

374.

375. begin

376. null;

377. end Start;

378.

379. begin

380. Put\_Line ("Lab5 started");

381. Start;

382. Put\_Line ("Lab5 finished");

383. END Pro2CW;

1. Код скрипта для тестування ПРГ1

1. import time, subprocess, sys

2.

3. COMMAND = ' '.join(sys.argv[1:])

4.

5. def measure\_time(affinity, command):

6. start\_t = time.time()

7.

8. subp = subprocess.Popen('start /AFFINITY 0x%X %s' % (pow(2, affinity) - 1, command), shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)

9. stderr, stdout = subp.communicate()

10. return\_code = subp.returncode

11. subp.wait()

12.

13. end\_t = time.time()

14.

15. return (end\_t - start\_t)

16.

17. for i in xrange(1, 5):

18. print 'Running %s with %d processors\n' % (COMMAND, i)

19. time\_elapsed = measure\_time(i, COMMAND)

20. print 'Time elapsed: %f' % time\_elapsed