**ДОДАТОК А. БЛОК-СХЕМА АЛГОРИТМУ ОСНОВНОЇ ПРОЦЕДУРИ ДЛЯ СИСТЕМИ ЗІ СПІЛЬНОЮ ПАМ’ЯТТЮ**

**ДОДАТОК Б. БЛОК-СХЕМА АЛГОРИТМУ РОБОТИ ПРОЦЕСІВ ДЛЯ СИСТЕМИ ЗІ СПІЛЬНОЮ ПАМ’ЯТТЮ**

**ДОДАТОК В. ЛІСТИНГ ПРОГРАМИ НА АДІ ДЛЯ СИСТЕМИ ЗІ СПІЛЬНОЮ ПАМ’ЯТТЮ**

1. //----------------------------------------------------------------------

2. //-- CurseWork 1 --

3. //-- C++.Win32 --

4. //-- Kostenchuk Volodymyr, IO-01 --

5. //-- MAh=(MB\*MCh)+alpha\*(MD\*MEh)+MUh --

6. //-- 28.02.2013 --

7. //-------------------------------------------------------------------------

8.

9. #include "stdafx.h"

10. #include "windows.h"

11. #include <iostream>

12. #include <fstream>

13. #include <algorithm>

14. #include <string>

15. #include <time.h>

16.

17. using namespace std;

18.

19. clock\_t TimeStart;

20. int N =2400;

21. const int P = 6;

22. const int H = N/P;

23.

24. //========================================================

25. char alpha;

26. char \*\* MB = new char\*[N];

27. char \*\* MC = new char\*[N];

28. char \*\* MD = new char\*[N];

29. char \*\* ME = new char\*[N];

30. char \*\* MU = new char\*[N];

31. char \*\* MA = new char\*[N];

32.

33.

34. HANDLE Sem\_in2,Sem\_in3,Sem\_in6;

35.

36. CRITICAL\_SECTION CS\_copy;

37.

38. HANDLE Mut;

39.

40. HANDLE Ev1, Ev3,Ev4,Ev5,Ev6;

41.

42. static void inputVector(char\* vec) {

43. for (int j = 0; j < N; j++) {

44. vec[j] = 1;

45. }

46. }

47.

48.

49. static void inputMatrix(char\*\* matr) {

50. for(int i=0; i<N; i++)

51. {

52. matr[i] = new char[N];

53. }

54.

55. for (int i = 0; i <N; i++) {

56. for (int j = 0; j <N; j++) {

57. matr[i][j] = 1;

58. }

59. }

60. }

61.

63. //================Задача\_1=======================

64.

65. void T1() {

66.

67. char \*\* MB1 = NULL;

68. char \*\* MD1 = NULL;

69.

70. MB1 = new char\*[N];

71. MC = new char\*[N];

72. MD1 = new char\*[N];

73. ME = new char\*[N];

74. MU = new char\*[N];

75. char alpha1;

76. for (int i=0; i<N; i++){

77.

78. MB1[i] = new char[N];

79. MD1[i] = new char[N];

80. MC[i] = new char[N];

81. ME[i] = new char[N];

82. MU[i] = new char[N];

83. }

84.

85. int task = 1;

86. cout << "T1 started" << endl;

87.

88. WaitForSingleObject(Sem\_in2, INFINITE);

89. WaitForSingleObject(Sem\_in3, INFINITE);

90. WaitForSingleObject(Sem\_in6, INFINITE);

91.

92. EnterCriticalSection(&CS\_copy);

93. for(int i = 0; i < N; i++){

94. for(int j = 0; j < N; j++){

95. MD1[i][j] = MD[i][j];

96. MB1[i][j] = MB[i][j];

97. }

98. }

99. LeaveCriticalSection(&CS\_copy);

100.

101. WaitForSingleObject(Mut, INFINITE);

102. alpha1 = alpha;

103. ReleaseMutex(Mut);

104.

105. for (int i=0; i<H; i++){

106. char sum;

107. char sum1;

108. for (int j=0; j<N; j++){

109.

110. sum=0;

111. sum1=0;

112. for (int k=0; k<N; k++){

113. sum =(sum+ MB1[k][j]\*MC[i][k] + alpha1 \* (MD1[k][j]\*ME[i][k])) ;

114. sum1=sum+ MU[i][k];

115. }

116.

117. MA[i][j]=sum1;

118. }

119. }

120.

121. SetEvent(Ev1);

122.

123. cout << "T1 finished" << endl;

124.

125. }

126.

127. //================Задача\_2===========================

128. void T2() {

129.

130. char \*\* MB2 = NULL;

131. char \*\* MD2 = NULL;

132.

133. MB2 = new char\*[N];

134. MC = new char\*[N];

135. MD2 = new char\*[N];

136. ME = new char\*[N];

137. MU = new char\*[N];

138. char alpha2;

139. for (int i=0; i<N; i++){

140. MA[i]=new char[N];

141. MB2[i] = new char[N];

142. MD2[i] = new char[N];

143. MC[i] = new char[N];

144. ME[i] = new char[N];

145. MU[i] = new char[N];

146. }

147.

148. int task = 2;

149. cout << "T2: started" << endl;

150. inputMatrix(MB);

151.

152. ReleaseSemaphore(Sem\_in2, 5, NULL);

153. WaitForSingleObject(Sem\_in3, INFINITE);

154. WaitForSingleObject(Sem\_in6, INFINITE);

155.

156. EnterCriticalSection(&CS\_copy);

157. for(int i = 0; i < N; i++){

158. for(int j = 0; j < N; j++){

159. MD2[i][j] = MD[i][j];

160. MB2[i][j] = MB[i][j];

161. }

162. }

163. LeaveCriticalSection(&CS\_copy);

164.

165. WaitForSingleObject(Mut, INFINITE);

166. alpha2 = alpha;

167. ReleaseMutex(Mut);

168.

169. for (int i=H; i<2\*H; i++){

170. char sum;

171. char sum1;

172. for (int j=0; j<N; j++){

173.

174. sum=0;

175. sum1=0;

176. for (int k=0; k<N; k++){

177. sum =(sum+ MB2[k][j]\*MC[i][k] + alpha2 \* (MD2[k][j]\*ME[i][k])) ;

178. sum1=sum+ MU[i][k];

179. }

180.

181. MA[i][j]=sum1;

182. }

183. }

184.

185. WaitForSingleObject(Ev1, INFINITE);

186. WaitForSingleObject(Ev3, INFINITE);

187. WaitForSingleObject(Ev4, INFINITE);

188. WaitForSingleObject(Ev5, INFINITE);

189. WaitForSingleObject(Ev6, INFINITE);

190.

191.

192. if (N<=13) {

193. for (int i = 0; i < N; i++) {

194. for (int j = 0; j < N; j++) {

195.

196. cout << (int)MA[i][j];

197. cout <<" ";

198.

199. }

200. cout <<" "<< endl;

201. }

202. }

203. cout << "T2 finished" << endl;

204. clock\_t TimeEnd=clock();

205. double tmp=(TimeEnd-TimeStart)/CLOCKS\_PER\_SEC;

206. cout <<"Time :";

207. cout << tmp << endl;

208.

209. }

210.

211. //================Задача\_3=======================

212. void T3() {

213.

214. char \*\* MB3 = NULL;

215. char \*\* MD3 = NULL;

216.

217. MB3 = new char\*[N];

218. MC = new char\*[N];

219. MD3 = new char\*[N];

220. ME = new char\*[N];

221. MU = new char\*[N];

222. char alpha3;

223.

224. for (int i=0; i<N; i++){

225. MB3[i] = new char[N];

226. MD3[i] = new char[N];

227. MC[i] = new char[N];

228. ME[i] = new char[N];

229. MU[i] = new char[N];

230. }

231.

232.

233. int task = 3;

234. cout << "T3: started" << endl;

235.

236. inputMatrix(MC);

237. inputMatrix(MD);

238. alpha = 1;

239.

240. ReleaseSemaphore(Sem\_in3, 5, NULL);

241. WaitForSingleObject(Sem\_in2, INFINITE);

242. WaitForSingleObject(Sem\_in6, INFINITE);

243.

244. EnterCriticalSection(&CS\_copy);

245. for(int i = 0; i < N; i++){

246. for(int j = 0; j < N; j++){

247. MD3[i][j] = MD[i][j];

248. MB3[i][j] = MB[i][j];

249. }

250. }

251. LeaveCriticalSection(&CS\_copy);

252.

253. WaitForSingleObject(Mut, INFINITE);

254. alpha3 = alpha;

255. ReleaseMutex(Mut);

256.

257. for (int i=2\*H; i<3\*H; i++){

258. char sum;

259. char sum1;

260. for (int j=0; j<N; j++){

261.

262. sum=0;

263. sum1=0;

264. for (int k=0; k<N; k++){

265. sum =(sum+ MB3[k][j]\*MC[i][k] + alpha3 \* (MD3[k][j]\*ME[i][k])) ;

266. sum1=sum+ MU[i][k];

267. }

268.

269. MA[i][j]=sum1;

270. }

271. }

272.

273. SetEvent(Ev3);

274.

275. cout << "T3 finished" << endl;

276. }

277. //================Задача\_4=======================

278. void T4() {

279.

280. char \*\* MB4 = NULL;

281. char \*\* MD4 = NULL;

282.

283. MB4 = new char\*[N];

284. MC = new char\*[N];

285. MD4 = new char\*[N];

286. ME = new char\*[N];

287. MU = new char\*[N];

288. char alpha4;

289. for (int i=0; i<N; i++){

290. MB4[i] = new char[N];

291. MD4[i] = new char[N];

292. MC[i] = new char[N];

293. ME[i] = new char[N];

294. MU[i] = new char[N];

295. }

296.

297. int task = 4;

298. cout << "T4: started" << endl;

299.

300. WaitForSingleObject(Sem\_in2, INFINITE);

301. WaitForSingleObject(Sem\_in3, INFINITE);

302. WaitForSingleObject(Sem\_in6, INFINITE);

303.

304. EnterCriticalSection(&CS\_copy);

305. for(int i = 0; i < N; i++){

306. for(int j = 0; j < N; j++){

307. MD4[i][j] = MD[i][j];

308. MB4[i][j] = MB[i][j];

309. }

310. }

311. LeaveCriticalSection(&CS\_copy);

312.

313. WaitForSingleObject(Mut, INFINITE);

314. alpha4 = alpha;

315. ReleaseMutex(Mut);

316.

317. for (int i=3\*H; i<4\*H; i++){

318. char sum;

319. char sum1;

320. for (int j=0; j<N; j++){

321.

322. sum=0;

323. sum1=0;

324. for (int k=0; k<N; k++){

325. sum =(sum+ MB4[k][j]\*MC[i][k] + alpha4 \* (MD4[k][j]\*ME[i][k])) ;

326. sum1=sum+ MU[i][k];

327. }

328.

329. MA[i][j]=sum1;

330. }

331. }

332.

333. SetEvent(Ev4);

334.

335. cout << "T4 finished" << endl;

336. }

337. //================Задача\_5=======================

338. void T5() {

339.

340. char \*\* MB5 = NULL;

341. char \*\* MD5 = NULL;

342.

343. MB5 = new char\*[N];

344. MC = new char\*[N];

345. MD5 = new char\*[N];

346. ME = new char\*[N];

347. MU = new char\*[N];

348. char alpha5;

349. for (int i=0; i<N; i++){

350. MB5[i] = new char[N];

351. MD5[i] = new char[N];

352. MC[i] = new char[N];

353. ME[i] = new char[N];

354. MU[i] = new char[N];

355. }

356.

357. int task = 5;

358. cout << "T5: started" << endl;

359.

360. WaitForSingleObject(Sem\_in2, INFINITE);

361. WaitForSingleObject(Sem\_in3, INFINITE);

362. WaitForSingleObject(Sem\_in6, INFINITE);

363.

364. EnterCriticalSection(&CS\_copy);

365. for(int i = 0; i < N; i++){

366. for(int j = 0; j < N; j++){

367. MD5[i][j] = MD[i][j];

368. MB5[i][j] = MB[i][j];

369. }

370. }

371. LeaveCriticalSection(&CS\_copy);

372.

373. WaitForSingleObject(Mut, INFINITE);

374. alpha5 = alpha;

375. ReleaseMutex(Mut);

376.

377. for (int i=4\*H; i<5\*H; i++){

378. char sum;

379. char sum1;

380. for (int j=0; j<N; j++){

381.

382. sum=0;

383. sum1=0;

384. for (int k=0; k<N; k++){

385. sum =(sum+ MB5[k][j]\*MC[i][k] + alpha5 \* (MD5[k][j]\*ME[i][k])) ;

386. sum1=sum+ MU[i][k];

387. }

388.

389. MA[i][j]=sum1;

390. }

391. }

392.

393. SetEvent(Ev5);

394.

395. cout << "T5 finished" << endl;

396. }

397. //================Задача\_6=======================

398. void T6() {

399. char \*\* MB6 = NULL;

400. char \*\* MD6 = NULL;

401.

402. MB6 = new char\*[N];

403. MC = new char\*[N];

404. MD6 = new char\*[N];

405. ME = new char\*[N];

406. MU = new char\*[N];

407. char alpha6;

408. for (int i=0; i<N; i++){

409. MB6[i] = new char[N];

410. MD6[i] = new char[N];

411. MC[i] = new char[N];

412. ME[i] = new char[N];

413. MU[i] = new char[N];

414. }

415.

416. int task = 6;

417. cout << "T6: started" << endl;

418.

419.

420. inputMatrix(MU);

421. inputMatrix(ME);

422.

423. ReleaseSemaphore(Sem\_in6, 5, NULL);

424. WaitForSingleObject(Sem\_in2, INFINITE);

425. WaitForSingleObject(Sem\_in3, INFINITE);

426.

427. EnterCriticalSection(&CS\_copy);

428. for(int i = 0; i < N; i++){

429. for(int j = 0; j < N; j++){

430. MD6[i][j] = MD[i][j];

431. MB6[i][j] = MB[i][j];

432. }

433. }

434. LeaveCriticalSection(&CS\_copy);

435.

436. WaitForSingleObject(Mut, INFINITE);

437. alpha6 = alpha;

438. ReleaseMutex(Mut);

439.

440. for (int i=5\*H; i<N; i++){

441. char sum;

442. char sum1;

443. for (int j=0; j<N; j++){

444.

445. sum=0;

446. sum1=0;

447. for (int k=0; k<N; k++){

448. sum =(sum+ MB6[k][j]\*MC[i][k] + alpha6 \* (MD6[k][j]\*ME[i][k])) ;

449. sum1=sum+ MU[i][k];

450. }

451.

452. MA[i][j]=sum1;

453. }

454. }

455.

456. SetEvent(Ev6);

457.

458. cout << "T6 finished" << endl;

459. }

460.

461.

462. //========================================================

463. int main(){

464. TimeStart=clock();

465. char a;

466. DWORD Tid1, Tid2, Tid3,Tid4,Tid5,Tid6;

467. HANDLE Thread1, Thread2, Thread3, Thread4, Thread5, Thread6;

468. cout << "Program started with H = " << H << endl;

469.

470. Ev1 = CreateEvent(NULL, 0, 0, NULL);

471. Ev3 = CreateEvent(NULL, 0, 0, NULL);

472. Ev4 = CreateEvent(NULL, 0, 0, NULL);

473. Ev5 = CreateEvent(NULL, 0, 0, NULL);

474. Ev6 = CreateEvent(NULL, 0, 0, NULL);

475.

476. Mut = CreateMutex(NULL, FALSE, NULL);

477.

478. Sem\_in2 = CreateSemaphore(NULL, 0, 5, NULL);

479. Sem\_in3 = CreateSemaphore(NULL, 0, 5, NULL);

480. Sem\_in6 = CreateSemaphore(NULL, 0, 5, NULL);

481.

482. InitializeCriticalSection(&CS\_copy);

483.

484. size\_t st = 50 \* 1024 \* 1024;

485.

486. Thread1 = CreateThread(NULL,0,(LPTHREAD\_START\_ROUTINE) T1,NULL,0,&Tid1);

487. Thread2 = CreateThread(NULL,0,(LPTHREAD\_START\_ROUTINE) T2,NULL,0,&Tid2);

488. Thread3 = CreateThread(NULL,0,(LPTHREAD\_START\_ROUTINE) T3,NULL,0,&Tid3);

489. Thread4 = CreateThread(NULL,0,(LPTHREAD\_START\_ROUTINE) T4,NULL,0,&Tid4);

490. Thread5 = CreateThread(NULL,0,(LPTHREAD\_START\_ROUTINE) T5,NULL,0,&Tid5);

491. Thread6 = CreateThread(NULL,0,(LPTHREAD\_START\_ROUTINE) T6,NULL,0,&Tid6);

492.

493. CloseHandle(Thread1);

494. CloseHandle(Thread2);

495. CloseHandle(Thread3);

496. CloseHandle(Thread4);

497. CloseHandle(Thread5);

498. CloseHandle(Thread6);

499.

500. cin.get();

501. return 0;

502. }

**ДОДАТОК г. БЛОК-СХЕМА АЛГОРИТМУ ОСНОВНОЇ ПРОЦЕДУРИ ДЛЯ СИСТЕМИ ЗІ ЛОКАЛЬНОЮ ПАМ’ЯТТЮ**

**ДОДАТОК Д. БЛОК-СХЕМА АЛГОРИТМУ РОБОТИ ПРОЦЕСІВ ДЛЯ СИСТЕМИ ЗІ ЛОКАЛЬНОЮ ПАМ’ЯТТЮ**

**ДОДАТОК Е. ЛІСТИНГ ПРОГРАМИ ДЛЯ СИСТЕМИ ЗІ ЛОКАЛЬНОЮ ПАМ’ЯТТЮ**

1. //-------------------------------------------------------------------------

2. //-- CurseWork 2 --

3. //-- C++. MPI --

4. //-- Kostenchuk Volodymyr, IO-01 --

5. //-- MAh=(MBh\*MC)+alpha\*(MDh\*ME)+MUh --

6. //-- 17.04.2013 --

7. //--------------------------------------------------------------------------

8.

9. #include <iostream>

10. #include <fstream>

11. #include <mpi.h>

12. #include <Windows.h>

13. #include "matrix.h"

14.

15.

16. using namespace std;

17.

18. const size\_t T1 = 0;

19. const size\_t T2 = 1;

20. const size\_t T3 = 2;

21. const size\_t T4 = 3;

22. const size\_t T5 = 4;

23. const size\_t T6 = 5;

24.

25. size\_t N;

26. size\_t P;

27. size\_t H;

28.

29. Matrix \*MC;

30. Matrix \*ME;

31. Matrix \*MA;

32. Matrix \*MB;

33. Matrix \*MD;

34. Matrix \*MU;

35.

36.

37. int alpha;

38. char\* fileName;

39.

40. LARGE\_INTEGER performanceFrequency;

41. LARGE\_INTEGER startTime;

42. LARGE\_INTEGER endTime;

43.

44.

45. int main(int args, char\* argv[]) {

46.

47. N = 2400;

48. P = 6;

49. H = N / P;

50.

51. MPI\_Init(&args, &argv);

52. int rank, size;

53. MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

54. MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

55.

56. size\_t sizeMa = 0,

57. sizeMb = 0,

58. sizeMd = 0,

59. sizeMu = 0;

60.

61. switch (rank) {

62. case T1:

63. sizeMa = H;

64. sizeMb = H;

65. sizeMd = H;

66. sizeMu = H;

67. break;

68. case T2:

69. sizeMa = N;

70. sizeMb = N;

71. sizeMd = N;

72. sizeMu = N;

73. break;

74. case T3:

75. sizeMa = H;

76. sizeMb = H;

77. sizeMd = H;

78. sizeMu = H;

79. break;

80. case T4:

81. sizeMa = 2\*H;

82. sizeMb = 2\*H;

83. sizeMd = 2\*H;

84. sizeMu = 2\*H;

85. break;

86. case T5:

87. sizeMa = H;

88. sizeMb = H;

89. sizeMd = H;

90. sizeMu = H;

91. break;

92. case T6:

93. sizeMa = H;

94. sizeMb = H;

95. sizeMd = H;

96. sizeMu = H;

97. break;

98. }

99.

100. void \* alphaPtr = &alpha;

101.

102. MC = new Matrix(N);

103. ME = new Matrix(N);

104. MA = new Matrix(sizeMa, N);

105. MB= new Matrix(sizeMb, N);

106. MD = new Matrix(sizeMd, N);

107. MU = new Matrix(sizeMu, N);

108.

109.

110. if(rank == T2) {

111. QueryPerformanceFrequency(&performanceFrequency);

112. QueryPerformanceCounter(&startTime);

113. }

114.

115. cout << "Task " << rank + 1 << " started" << endl;

116.

117. MPI\_Status stat;

118. switch (rank) {

119. case T1:

120.

121. MPI\_Recv(MB->get(0), H \* N, MPI\_INT, T2, 0, MPI\_COMM\_WORLD, &stat);

122. MPI\_Recv(MD->get(0), H \* N, MPI\_INT, T2, 0, MPI\_COMM\_WORLD, &stat);

123. MPI\_Recv(MU->get(0), H \* N, MPI\_INT, T2, 0, MPI\_COMM\_WORLD, &stat);

124. MPI\_Recv(alphaPtr, 1, MPI\_INT, T2, 0, MPI\_COMM\_WORLD, &stat);

125.

126. MPI\_Recv(ME->get(0), N \* N, MPI\_INT, T2, 0, MPI\_COMM\_WORLD, &stat);

127. MPI\_Recv(MC->get(0), N \* N, MPI\_INT, T2, 0, MPI\_COMM\_WORLD, &stat);

128.

129.

130. for (int i=0; i<H; i++){

131. char sum;

132. char sum1;

133. for (int j=0; j<N; j++){

134.

135. sum=0;

136. sum1=0;

137. for (int k=0; k<N; k++){

138. sum =(sum+ MB->get(i,k)\*MC->get(k,j) + (alpha\*(MD->get(i,k)\*ME->get(k,j)))) ;

139. sum1=sum+ MU->get(i,k);

140. }

141. MA->set(i,j,sum1);

142. }

143. }

144.

145. MPI\_Send(MA->get(0), H \* N, MPI\_INT, T2, 0, MPI\_COMM\_WORLD);

146. break;

147. case T2:

148. alpha = 1;

149. MB->fill(1);

150. MC->fill(1);

151. MU->fill(1);

152. MD->fill(1);

153. ME->fill(1);

154.

155. MPI\_Send(MB->get(0), H \* N, MPI\_INT, T1, 0, MPI\_COMM\_WORLD);

156. MPI\_Send(MD->get(0), H \* N, MPI\_INT, T1, 0, MPI\_COMM\_WORLD);

157. MPI\_Send(MU->get(0), H \* N, MPI\_INT, T1, 0, MPI\_COMM\_WORLD);

158. MPI\_Send(alphaPtr, 1, MPI\_INT, T1, 0, MPI\_COMM\_WORLD);

159.

160. MPI\_Send(ME->get(0), N \* N, MPI\_INT, T1, 0, MPI\_COMM\_WORLD);

161. MPI\_Send(MC->get(0), N \* N, MPI\_INT, T1, 0, MPI\_COMM\_WORLD);

162.

163.

164. MPI\_Send(MB->get(2\*H), H \* N, MPI\_INT, T3, 0, MPI\_COMM\_WORLD);

165. MPI\_Send(MD->get(2\*H), H \* N, MPI\_INT, T3, 0, MPI\_COMM\_WORLD);

166. MPI\_Send(MU->get(2\*H), H \* N, MPI\_INT, T3, 0, MPI\_COMM\_WORLD);

167. MPI\_Send(alphaPtr, 1, MPI\_INT, T3, 0, MPI\_COMM\_WORLD);

168.

169. MPI\_Send(ME->get(2\*H), N \* N, MPI\_INT, T3, 0, MPI\_COMM\_WORLD);

170. MPI\_Send(MC->get(2\*H), N \* N, MPI\_INT, T3, 0, MPI\_COMM\_WORLD);

171.

172.

173. MPI\_Send(MB->get(3\*H), H \* N, MPI\_INT, T4, 0, MPI\_COMM\_WORLD);

174. MPI\_Send(MD->get(3\*H), H \* N, MPI\_INT, T4, 0, MPI\_COMM\_WORLD);

175. MPI\_Send(MU->get(3\*H), H \* N, MPI\_INT, T4, 0, MPI\_COMM\_WORLD);

176. MPI\_Send(alphaPtr, 1, MPI\_INT, T4, 0, MPI\_COMM\_WORLD);

177.

178. MPI\_Send(MC->get(3\*H), N \* N, MPI\_INT, T4, 0, MPI\_COMM\_WORLD);

179. MPI\_Send(ME->get(3\*H), N \* N, MPI\_INT, T4, 0, MPI\_COMM\_WORLD);

180.

181. MPI\_Send(MB->get(4\*H), H \* N, MPI\_INT, T5, 0, MPI\_COMM\_WORLD);

182. MPI\_Send(MC->get(4\*H), N \* N, MPI\_INT, T5, 0, MPI\_COMM\_WORLD);

183. MPI\_Send(MD->get(4\*H), H \* N, MPI\_INT, T5, 0, MPI\_COMM\_WORLD);

184. MPI\_Send(ME->get(4\*H), N \* N, MPI\_INT, T5, 0, MPI\_COMM\_WORLD);

185. MPI\_Send(MU->get(4\*H), H \* N, MPI\_INT, T5, 0, MPI\_COMM\_WORLD);

186. MPI\_Send(alphaPtr, 1, MPI\_INT, T5, 0, MPI\_COMM\_WORLD);

187.

188. MPI\_Send(MB->get(5\*H), H \* N, MPI\_INT, T4, 0, MPI\_COMM\_WORLD);

189. MPI\_Send(MC->get(5\*H), N \* N, MPI\_INT, T4, 0, MPI\_COMM\_WORLD);

190. MPI\_Send(MD->get(5\*H), H \* N, MPI\_INT, T4, 0, MPI\_COMM\_WORLD);

191. MPI\_Send(ME->get(5\*H), N \* N, MPI\_INT, T4, 0, MPI\_COMM\_WORLD);

192. MPI\_Send(MU->get(5\*H), H \* N, MPI\_INT, T4, 0, MPI\_COMM\_WORLD);

193.

194.

195. for (int i=0; i<H; i++){

196. char sum;

197. char sum1;

198. for (int j=0; j<N; j++){

199.

200. sum=0;

201. sum1=0;

202. for (int k=0; k<N; k++){

203. sum =(sum+ MB->get(i,k)\*MC->get(k,j) + (alpha\*(MD->get(i,k)\*ME->get(k,j)))) ;

204. sum1=sum+ MU->get(i,k);

205. }

206. MA->set(i,j,sum1);

207. }

208. }

209.

210. MPI\_Recv(MA->get(0), H \* N, MPI\_INT, T1, 0, MPI\_COMM\_WORLD, &stat);

211. MPI\_Recv(MA->get(2\*H), H \* N, MPI\_INT, T3, 0, MPI\_COMM\_WORLD, &stat);

212. MPI\_Recv(MA->get(4\*H), H \* N, MPI\_INT, T5, 0, MPI\_COMM\_WORLD, &stat);

213. MPI\_Recv(MA->get(3\*H), 2\*H \* N, MPI\_INT, T4, 0, MPI\_COMM\_WORLD, &stat);

214.

215. if(MA->cols < 13) {

216. cout << \*MA << endl;

217. }

218. break;

219.

220. case T3:

221.

222. MPI\_Recv(MD->get(0), N \* N, MPI\_INT, T2, 0, MPI\_COMM\_WORLD, &stat);

223. MPI\_Recv(MB->get(0), H \* N, MPI\_INT, T2, 0, MPI\_COMM\_WORLD, &stat);

224. MPI\_Recv(MU->get(0), H \* N, MPI\_INT, T2, 0, MPI\_COMM\_WORLD, &stat);

225. MPI\_Recv(alphaPtr, 1, MPI\_INT, T2, 0, MPI\_COMM\_WORLD, &stat);

226.

227. MPI\_Recv(ME->get(0), N \* N, MPI\_INT, T2, 0, MPI\_COMM\_WORLD, &stat);

228. MPI\_Recv(MC->get(0), N \* N, MPI\_INT, T2, 0, MPI\_COMM\_WORLD, &stat);

229.

230.

231. for (int i=0; i<H; i++){

232. char sum;

233. char sum1;

234. for (int j=0; j<N; j++){

235.

236. sum=0;

237. sum1=0;

238. for (int k=0; k<N; k++){

239. sum =(sum+ MB->get(i,k)\*MC->get(k,j) + (alpha\*(MD->get(i,k)\*ME->get(k,j)))) ;

240. sum1=sum+ MU->get(i,k);

241. }

242. MA->set(i,j,sum1);

243. }

244. }

245.

246. MPI\_Send(MA->get(0), H \* N, MPI\_INT, T2, 0, MPI\_COMM\_WORLD);

247. break;

248. case T4:

249.

250. MPI\_Recv(MB->get(0), H \* N, MPI\_INT, T2, 0, MPI\_COMM\_WORLD, &stat);

251. MPI\_Recv(MD->get(0), H \* N, MPI\_INT, T2, 0, MPI\_COMM\_WORLD, &stat);

252. MPI\_Recv(MU->get(0), H \* N, MPI\_INT, T2, 0, MPI\_COMM\_WORLD, &stat);

253. MPI\_Recv(alphaPtr, 1, MPI\_INT, T2, 0, MPI\_COMM\_WORLD, &stat);

254.

255. MPI\_Recv(MC->get(0), N \* N, MPI\_INT, T2, 0, MPI\_COMM\_WORLD, &stat);

256. MPI\_Recv(ME->get(0), N \* N, MPI\_INT, T2, 0, MPI\_COMM\_WORLD, &stat);

257.

258.

259. MPI\_Recv(MB->get(H), H \* N, MPI\_INT, T2, 0, MPI\_COMM\_WORLD, &stat);

260. MPI\_Recv(MC->get(H), N \* N, MPI\_INT, T2, 0, MPI\_COMM\_WORLD, &stat);

261. MPI\_Recv(MD->get(H), H \* N, MPI\_INT, T2, 0, MPI\_COMM\_WORLD, &stat);

262. MPI\_Recv(ME->get(H), N \* N, MPI\_INT, T2, 0, MPI\_COMM\_WORLD, &stat);

263. MPI\_Recv(MU->get(H), H \* N, MPI\_INT, T2, 0, MPI\_COMM\_WORLD, &stat);

264.

265.

266. // Пересилка задачі Т6

267. MPI\_Send(MB->get(H), H \* N, MPI\_INT, T6, 0, MPI\_COMM\_WORLD);

268. MPI\_Send(MC->get(H), N \* N, MPI\_INT, T6, 0, MPI\_COMM\_WORLD);

269. MPI\_Send(MD->get(H), H \* N, MPI\_INT, T6, 0, MPI\_COMM\_WORLD);

270. MPI\_Send(ME->get(H), N \* N, MPI\_INT, T6, 0, MPI\_COMM\_WORLD);

271. MPI\_Send(MU->get(H), H \* N, MPI\_INT, T6, 0, MPI\_COMM\_WORLD);

272. MPI\_Send(alphaPtr, 1, MPI\_INT, T6, 0, MPI\_COMM\_WORLD);

273.

274. for (int i=0; i<H; i++){

275. char sum;

276. char sum1;

277. for (int j=0; j<N; j++){

278.

279. sum=0;

280. sum1=0;

281. for (int k=0; k<N; k++){

282. sum =(sum+ MB->get(k,j)\*MC->get(i,k) + alpha \* (MD->get(k,j)\*ME->get(i,k))) ;

283. sum1=sum+ MU->get(i,k);

284. }

285.

286. MA->set(i,j,sum1);

287. }

288. }

289. MPI\_Recv(MA->get(0), H \* N, MPI\_INT, T6, 0, MPI\_COMM\_WORLD, &stat);

290. MPI\_Send(MA->get(0), 2\*H \* N, MPI\_INT, T2, 0, MPI\_COMM\_WORLD);

291. break;

292.

293.

294. case T5:

295. MPI\_Recv(ME->get(0), N \* N, MPI\_INT, T2, 0, MPI\_COMM\_WORLD, &stat);

296. MPI\_Recv(MC->get(0), N \* N, MPI\_INT, T2, 0, MPI\_COMM\_WORLD, &stat);

297.

298. MPI\_Recv(MB->get(0), H \* N, MPI\_INT, T2, 0, MPI\_COMM\_WORLD, &stat);

299. MPI\_Recv(MD->get(0), H \* N, MPI\_INT, T2, 0, MPI\_COMM\_WORLD, &stat);

300. MPI\_Recv(MU->get(0), H \* N, MPI\_INT, T2, 0, MPI\_COMM\_WORLD, &stat);

301.

302. MPI\_Recv(alphaPtr, 1, MPI\_INT, T2, 0, MPI\_COMM\_WORLD, &stat);

303.

304.

305. for (int i=0; i<H; i++){

306. char sum;

307. char sum1;

308. for (int j=0; j<N; j++){

309.

310. sum=0;

311. sum1=0;

312. for (int k=0; k<N; k++){

313. sum =(sum+ MB->get(k,j)\*MC->get(i,k) + alpha \* (MD->get(k,j)\*ME->get(i,k))) ;

314. sum1=sum+ MU->get(i,k);

315. }

316.

317. MA->set(i,j,sum1);

318. }

319. }

320.

321. MPI\_Send(MA->get(0), H \* N, MPI\_INT, T2, 0, MPI\_COMM\_WORLD);

322. break;

323. case T6:

324. MPI\_Recv(MB->get(0), H \* N, MPI\_INT, T4, 0, MPI\_COMM\_WORLD, &stat);

325. MPI\_Recv(MC->get(0), N \* N, MPI\_INT, T4, 0, MPI\_COMM\_WORLD, &stat);

326. MPI\_Recv(ME->get(0), N \* N, MPI\_INT, T4, 0, MPI\_COMM\_WORLD, &stat);

327. MPI\_Recv(MD->get(0), H \* N, MPI\_INT, T4, 0, MPI\_COMM\_WORLD, &stat);

328. MPI\_Recv(MU->get(0), H \* N, MPI\_INT, T4, 0, MPI\_COMM\_WORLD, &stat);

329. MPI\_Recv(alphaPtr, 1, MPI\_INT, T4, 0, MPI\_COMM\_WORLD, &stat);

330.

331.

332. for (int i=0; i<H; i++){

333. char sum;

334. char sum1;

335. for (int j=0; j<N; j++){

336.

337. sum=0;

338. sum1=0;

339. for (int k=0; k<N; k++){

340. sum =(sum+ MB->get(k,j)\*MC->get(i,k) + alpha \* (MD->get(k,j)\*ME->get(i,k))) ;

341. sum1=sum+ MU->get(i,k);

342. }

343.

344. MA->set(i,j,sum1);

345. }

346. }

347.

348. MPI\_Send(MA->get(0), MA->rows\*MA->cols, MPI\_INT, T4, 0, MPI\_COMM\_WORLD);

349. break;

350. }

351.

352. cout << "Task " << rank + 1 << " finished" << endl;

353.

354. if(rank == T2) {

355.

356. QueryPerformanceCounter(&endTime);

357. double elapsedTime = ((double) (endTime.QuadPart - startTime.QuadPart))

358. / (double) performanceFrequency.QuadPart;

359. cout << "PRG2: computations took " << (int) (elapsedTime \* 1000) << " ms" << endl;

360.

361. ofstream os(fileName, ofstream::out | ofstream::app);

362. os << "N: " << N << "\tProcess time: "

363. << (int)(elapsedTime \* 1000) << "\tms ";

364. os.close();

365.

366. }

367. system("pause");

368. MPI\_Finalize();

369.

370. return 0;

371. }

|  |
| --- |
|  |

1. //----------------------------------------------

2. //-- CurseWork 2 --

3. //-- C++. MPI --

4. //-- Kostenchuk Volodymyr, IO-01 --

5. //-- MAh=(MBh\*MC)+alpha\*(MDh\*ME)+MUh --

6. //-- 17.04.2013 --

7. //----------------------------------------------

8.

9.

10. #include "matrix.h"

11.

12. Matrix::Matrix(size\_t N) :

13. rows(N),

14. cols(N),

15. data(new int[N \* N])

16. {

17. }

18.

19. Matrix::Matrix(size\_t rows, size\_t cols) :

20. rows(rows),

21. cols(cols),

22. data(new int[rows \* cols])

23. {

24. for(size\_t i = 0; i < this->rows; i++) {

25. for(size\_t j = 0; j < this->cols; j++) {

26. this->data[i \* this->cols + j] = 0;

27. }

28. }

29. }

30.

31. Matrix::Matrix(const Matrix &other) :

32. rows(other.rows),

33. cols(other.cols),

34. data(new int[other.rows \* other.cols])

35. {

36. for(size\_t i = 0; i < this->rows \* this->cols; i++) {

37. this->data[i] = other.data[i];

38. }

39. }

40.

41.

42. Matrix::~Matrix() {

43. delete[] data;

44. }

45.

46. void Matrix::fill(int value) {

47. for(size\_t i = 0; i < this->rows; i++) {

48. for(size\_t j = 0; j < this->cols; j++) {

49. this->data[i \* this->cols + j] = value;

50. }

51. }

52. }

53.

54.

55. std::ostream &operator<<(std::ostream &ostr, const Matrix &matrix) {

56. for(size\_t i = 0; i < matrix.rows; i++) {

57. for(size\_t j = 0; j < matrix.cols; j++) {

58. ostr << matrix.data[i \* matrix.cols + j] << " ";

59. }

60. ostr << std::endl;

61. }

62. return ostr;

63. }

|  |
| --- |
|  |

1. //----------------------------------------------

2. //-- CurseWork 2 --

3. //-- C++. MPI --

4. //-- Kostenchuk Volodymyr, IO-01 --

5. //-- MAh=(MBh\*MC)+alpha\*(MDh\*ME)+MUh --

6. //-- 17.04.2013 --

7. //----------------------------------------------

8.

9. #ifndef \_MATRIX\_H\_

10. #define \_MATRIX\_H\_ 1

11.

12. #include <ostream>

13. #include <assert.h>

14.

15. class Matrix {

16. public:

17. Matrix(size\_t rows, size\_t cols);

18. Matrix(size\_t N);

19. Matrix(Matrix const &other);

20. ~Matrix();

21.

22. int get(size\_t i, size\_t j) const {

23. assert(i < this->rows);

24. assert(j < this->cols);

25. return this->data[i \* cols + j];

26. }

27.

28. void set(size\_t i, size\_t j, int value) {

29. assert(i < this->rows);

30. assert(j < this->cols);

31. this->data[i \* cols + j] = value;

32. }

33.

34. void \* get(size\_t element) {

35. return this->data + element;

36. }

37.

38. void fill(int value);

39.

40. const size\_t rows;

41. const size\_t cols;

42.

43. private:

44. int \*data;

45. friend std::ostream &operator<<(std::ostream &ostr, const Matrix &m);

46.

47. };

48.

49. #endif