Міністерство освіти і науки, молоді та спорту України

Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

# Лабораторна робота №4

# з курсу «Паралельні та розподілені обчислення-2»

**Тема:** Паралельне програмування за допомогою OpenMP

Виконав: студент групи ІО-93

Глухенько Костянтин Анатолійович

2012 р.

**Завдання**

Математичний вираз: A=SORT(B\*(MO\*MZ)+C)

Засоби програмування: OpenMP (Критичні секції, бар’єри, замки, атомарність)

Структура ПКС:

ЗП

Т1

Т3

Т6

Т4

Т5

Т2

A B C

MO

MZ

**Виконання**

**Етап 1.** Побудова паралельного алгоритму.

1. AHS=SORT(B\*(MO\*MZH)+CH)
2. A=SORT(AHS, AHS, AHS, AHS, AHS, AHS)

Н=N/P

СР: B,C,MO

**Етап 2.** Розроблення алгоритмів роботи кожного процесу.

|  |  |  |
| --- | --- | --- |
|  | Т1 | Точки синхронізації |
| 1) | Введення MZ |  |
| 2) | Сигнал про введення MZ в задачі Т1 | S2,3,4,5,6-1 |
| 3) | Чекати введення МО в задачі Т2 | W2-1 |
| 4) | Чекати введення В і С в задачі Т6 | W6-1 |
| 5) | Копіювання МО1 = МО | КД |
| 6) | Копіювання В1 = В | КД |
| 7) | Обчислення AHS=SORT(B1\*(MO1\*MZH)+CH) |  |
| 8) | Сигнал про завершення обчислень AHS | S6-2 |

|  |  |  |
| --- | --- | --- |
|  | Т2 | Точки синхронізації |
| 1) | Введення MО |  |
| 2) | Чекати введення МZ в задачі Т1 | W1-1 |
| 3) | Сигнал про введення MО в задачі Т2 | S1,3,4,5,6-1 |
| 4) | Чекати введення В і С в задачі Т6 | W6-1 |
| 5) | Копіювання МО2 = МО | КД |
| 6) | Копіювання В2 = В | КД |
| 7) | Обчислення AHS=SORT(B2\*(MO2\*MZH)+CH) |  |
| 8) | Сигнал про завершення обчислень AHS | S6-2 |

|  |  |  |
| --- | --- | --- |
|  | Т3 | Точки синхронізації |
| 1) | Чекати на введення МZ в задачі Т1 | W1-1 |
| 2) | Чекати на введення МО в задачі Т2 | W2-1 |
| 3) | Чекати введення В і С в задачі Т6 | W6-1 |
| 4) | Копіювання МО3 = МО | КД |
| 5) | Копіювання В3 = В | КД |
| 6) | Обчислення AHS=SORT(B3\*(MO3\*MZH)+CH) |  |
| 7) | Сигнал про завершення обчислень AHS | S6-2 |

|  |  |  |
| --- | --- | --- |
|  | Т4 | Точки синхронізації |
| 1) | Чекати на введення МZ в задачі Т1 | W1-1 |
| 2) | Чекати на введення МО в задачі Т2 | W2-1 |
| 3) | Чекати введення В і С в задачі Т6 | W6-1 |
| 4) | Копіювання МО4 = МО | КД |
| 5) | Копіювання В4 = В | КД |
| 6) | Обчислення AHS=SORT(B4\*(MO4\*MZH)+CH) |  |
| 7) | Сигнал про завершення обчислень AHS | S6-2 |

|  |  |  |
| --- | --- | --- |
|  | Т5 | Точки синхронізації |
| 1) | Чекати на введення МZ в задачі Т1 | W1-1 |
| 2) | Чекати на введення МО в задачі Т2 | W2-1 |
| 3) | Чекати введення В і С в задачі Т6 | W6-1 |
| 4) | Копіювання МО5 = МО | КД |
| 5) | Копіювання В5 = В | КД |
| 6) | Обчислення AHS=SORT(B5\*(MO5\*MZH)+CH) |  |
| 7) | Сигнал про завершення обчислень AHS | S6-2 |

|  |  |  |
| --- | --- | --- |
|  | Т6 | Точки синхронізації |
| 1) | Введення В і С |  |
| 2) | Чекати введення МZ в задачі Т1 | W1-1 |
| 3) | Чекати введення МO в задачі Т2 | W2-1 |
| 4) | Сигнал про введення В і С в задачі Т6 | S1,2,3,4,5-1 |
| 5) | Копіювання МО2 = МО | КД |
| 6) | Копіювання В2 = В | КД |
| 7) | Обчислення AHS=SORT(B2\*(MO2\*MZH)+CH) |  |
| 8) | Чекати завершення обчислень AHS | W1,2,3,4,5-2 |
| 9) | Обчислення A=SORT(AHS, AHS, AHS, AHS, AHS, AHS) |  |
| 10) | Виведення А |  |

**Етап 3.** Розроблення структурної схеми взаємодії задач.

B1

Т2

Т1

Т4

Т3

Т5

Т6

B2

CS1

L1

На структурній схемі введені такі елементи:

B1 бар’єр для синхронізації із завершенням введення

B2 бар’єр для синхронізації із завершенням обчислень

L1 замок для керування доступом до спільного ресурсу (B)

CS1 критична секція для керування доступом до спільного ресурсу (MO)

**Етап 4.** Розроблення програми**.**

1. /\*Глухенько Костянтин Анатолійович, група ІО-93
2. лаб№ 4. Паралельне програмування за допомогою OpenMP
3. A=SORT(B\*(MO\*MZ)+C)\*/
4. #include "stdafx.h"
5. #include <iostream>
6. #include <fstream>
7. #include <omp.h>
8. using std::cout;
9. using std::endl;
10. int const N = 12;
11. int const P = 6;
12. int const H = N/P;
13. int const val = 1;
14. omp\_lock\_t L1;
15. struct Matrix{
16. int mas[N][N];
17. };
18. Matrix MO, MZ, MO1, MO2, MO3, MO4, MO5, MO6;
19. int As[N], A[N], B[N], C[N], B1[N], B2[N], B3[N], B4[N], B5[N], B6[N];
20. //====================================================================
21. void Thread\_T1\_B1(){
22. std:: cout << "T1 started" << std::endl;
23. //1
24. for(int i = 0; i < N; i++)
25. for(int j = 0; j < N; j++)
26. MZ.mas[i][j] = val;
27. }
28. void Thread\_T1\_B2(){
29. //5
30. #pragma omp critical
31. {
32. MO1 = MO;
33. }
34. //6
35. omp\_set\_lock(&L1);
36. for(int i = 0; i < N; i++)
37. B1[i] = B[i];
38. omp\_unset\_lock(&L1);
39. //7
40. for(int i = 0; i < H; i++){
41. for(int j = 0; j < N; j++){
42. int t = 0;
43. for(int k = 0; k < N; k++){
44. t += MO1.mas[k][j] \* MZ.mas[i][k];
45. }
46. As[i] += t\*B1[j];
47. }
48. As[i] += C[i];
49. }
50. for(int i = 0; i < H; i++)
51. for(int j = i+1; j < H; j++)
52. if(As[i] > As[j]){
53. int t = As[j];
54. As[j] = As[i];
55. As[i] = t;
56. }
57. std:: cout << "T1 started" << std::endl;
58. }
59. //====================================================================
60. void Thread\_T2\_B1(){
61. std:: cout << "T2 started" << std::endl;
62. //1
63. for(int i = 0; i < N; i++)
64. for(int j = 0; j < N; j++)
65. MO.mas[i][j] = val;
66. }
67. void Thread\_T2\_B2(){
68. //5
69. #pragma omp critical
70. {
71. MO2 = MO;
72. }
73. //6
74. omp\_set\_lock(&L1);
75. for(int i = 0; i < N; i++)
76. B2[i] = B[i];
77. omp\_unset\_lock(&L1);
78. //7
79. for(int i = H; i < H\*2; i++){
80. for(int j = 0; j < N; j++){
81. int t = 0;
82. for(int k = 0; k < N; k++){
83. t += MO2.mas[k][j] \* MZ.mas[i][k];
84. }
85. As[i] += t\*B2[j];
86. }
87. As[i] += C[i];
88. }
89. for(int i = H; i < H\*2; i++)
90. for(int j = i+1; j < H; j++)
91. if(As[i] > As[j]){
92. int t = As[j];
93. As[j] = As[i];
94. As[i] = t;
95. }
96. std:: cout << "T2 started" << std::endl;
97. }
98. //====================================================================
99. void Thread\_T3\_B1(){
100. std:: cout << "T3 started" << std::endl;
101. }
102. void Thread\_T3\_B2(){
103. //5
104. #pragma omp critical
105. {
106. MO3 = MO;
107. }
108. //6
109. omp\_set\_lock(&L1);
110. for(int i = 0; i < N; i++)
111. B3[i] = B[i];
112. omp\_unset\_lock(&L1);
113. //7
114. for(int i = H\*2; i < H\*3; i++){
115. for(int j = 0; j < N; j++){
116. int t = 0;
117. for(int k = 0; k < N; k++){
118. t += MO3.mas[k][j] \* MZ.mas[i][k];
119. }
120. As[i] += t\*B3[j];
121. }
122. As[i] += C[i];
123. }
124. for(int i = H\*2; i < H\*3; i++)
125. for(int j = i+1; j < H; j++)
126. if(As[i] > As[j]){
127. int t = As[j];
128. As[j] = As[i];
129. As[i] = t;
130. }
131. std:: cout << "T3 started" << std::endl;
132. }
133. //====================================================================
134. void Thread\_T4\_B1(){
135. std:: cout << "T4 started" << std::endl;
136. }
137. void Thread\_T4\_B2(){
138. //5
139. #pragma omp critical
140. {
141. MO4 = MO;
142. }
143. //6
144. omp\_set\_lock(&L1);
145. for(int i = 0; i < N; i++)
146. B4[i] = B[i];
147. omp\_unset\_lock(&L1);
148. //7
149. for(int i = H\*3; i < H\*4; i++){
150. for(int j = 0; j < N; j++){
151. int t = 0;
152. for(int k = 0; k < N; k++){
153. t += MO4.mas[k][j] \* MZ.mas[i][k];
154. }
155. As[i] += t\*B4[j];
156. }
157. As[i] += C[i];
158. }
159. for(int i = H\*3; i < H\*4; i++)
160. for(int j = i+1; j < H; j++)
161. if(As[i] > As[j]){
162. int t = As[j];
163. As[j] = As[i];
164. As[i] = t;
165. }
166. std:: cout << "T4 started" << std::endl;
167. }
168. //====================================================================
169. void Thread\_T5\_B1(){
170. std:: cout << "T5 started" << std::endl;
171. }
172. void Thread\_T5\_B2(){
173. //5
174. #pragma omp critical
175. {
176. MO5 = MO;
177. }
178. //6
179. omp\_set\_lock(&L1);
180. for(int i = 0; i < N; i++)
181. B5[i] = B[i];
182. omp\_unset\_lock(&L1);
183. //7
184. for(int i = H\*4; i < H\*5; i++){
185. for(int j = 0; j < N; j++){
186. int t = 0;
187. for(int k = 0; k < N; k++){
188. t += MO5.mas[k][j] \* MZ.mas[i][k];
189. }
190. As[i] += t\*B5[j];
191. }
192. As[i] += C[i];
193. }
194. for(int i = H\*4; i < H\*5; i++)
195. for(int j = i+1; j < H; j++)
196. if(As[i] > As[j]){
197. int t = As[j];
198. As[j] = As[i];
199. As[i] = t;
200. }
201. std:: cout << "T5 started" << std::endl;
202. }
203. //====================================================================
204. void Thread\_T6\_B1(){
205. std:: cout << "T6 started" << std::endl;
206. //1
207. for(int i = 0; i < N; i++){
208. B[i] = val;
209. C[i] = val;
210. }
211. }
212. void Thread\_T6\_B2(){
213. //5
214. #pragma omp critical
215. {
216. MO6 = MO;
217. }
218. //6
219. omp\_set\_lock(&L1);
220. for(int i = 0; i < N; i++)
221. B6[i] = B[i];
222. omp\_unset\_lock(&L1);
223. //7
224. for(int i = H\*5; i < H\*6; i++){
225. for(int j = 0; j < N; j++){
226. int t = 0;
227. for(int k = 0; k < N; k++){
228. t += MO6.mas[k][j] \* MZ.mas[i][k];
229. }
230. As[i] += t\*B6[j];
231. }
232. As[i] += C[i];
233. }
234. for(int i = H\*5; i < H\*6; i++)
235. for(int j = i+1; j < H; j++)
236. if(As[i] > As[j]){
237. int t = As[j];
238. As[j] = As[i];
239. As[i] = t;
240. }
241. }
242. void Thread\_T6\_B3(){
243. int t1 = 0;
244. int t2 = H;
245. int t3 = H\*2;
246. int t4 = H\*3;
247. int t5 = H\*4;
248. int t6 = H\*5;
249. for(int i = 0; i < N; i++){
250. if((As[t1] <= As[t2]) && (As[t1] <= As[t3]) && (As[t1] <= As[t4]) && (As[t1] <= As[t5]) && (As[t1] <= As[t6]) && (t1 < H)){
251. A[i] = As[t1];
252. t1++;
253. }
254. else if((As[t2] <= As[t1]) && (As[t2] <= As[t3]) && (As[t2] <= As[t4]) && (As[t2] <= As[t5]) && (As[t2] <= As[t6]) && (t2 < H\*2)){
255. A[i] = As[t2];
256. t2++;
257. }
258. else if((As[t3] <= As[t1]) && (As[t3] <= As[t2]) && (As[t3] <= As[t4]) && (As[t3] <= As[t5]) && (As[t3] <= As[t6]) && (t3 < H\*3)){
259. A[i] = As[t3];
260. t3++;
261. }
262. else if((As[t4] <= As[t1]) && (As[t4] <= As[t3]) && (As[t4] <= As[t2]) && (As[t4] <= As[t5]) && (As[t4] <= As[t6]) && (t4 < H\*4)){
263. A[i] = As[t4];
264. t4++;
265. }
266. else if((As[t5] <= As[t1]) && (As[t5] <= As[t3]) && (As[t5] <= As[t4]) && (As[t5] <= As[t2]) && (As[t5] <= As[t6]) && (t5 < H\*5)){
267. A[i] = As[t5];
268. t5++;
269. }
270. else if((As[t6] <= As[t1]) && (As[t6] <= As[t3]) && (As[t6] <= As[t4]) && (As[t6] <= As[t2]) && (As[t6] <= As[t5]) && (t6 < H\*6)){
271. A[i] = As[t6];
272. t6++;
273. }
274. }
275. cout << "A[0] = "<< A[0] << endl;
276. }
277. //====================================================================
278. int \_tmain(int argc, \_TCHAR\* argv[])
279. {
280. cout << "lab started"<< endl;
281. omp\_set\_num\_threads(P);
282. omp\_init\_lock(&L1);
283. #pragma comment(linker, "/STACK:599999999")
284. #pragma omp parallel
285. {
286. switch (omp\_get\_thread\_num()){
287. case 0: Thread\_T1\_B1();
288. break;
289. case 1: Thread\_T2\_B1();
290. break;
291. case 2: Thread\_T3\_B1();
292. break;
293. case 3: Thread\_T4\_B1();
294. break;
295. case 4: Thread\_T5\_B1();
296. break;
297. case 5: Thread\_T6\_B1();
298. break;
299. }
300. #pragma omp barrier //B1
301. switch (omp\_get\_thread\_num()){
302. case 0: Thread\_T1\_B2();
303. break;
304. case 1: Thread\_T2\_B2();
305. break;
306. case 2: Thread\_T3\_B2();
307. break;
308. case 3: Thread\_T4\_B2();
309. break;
310. case 4: Thread\_T5\_B2();
311. break;
312. case 5: Thread\_T6\_B2();
313. break;
314. }
315. #pragma omp barrier //B2
316. if (omp\_get\_thread\_num() == 5)
317. Thread\_T6\_B3();
318. }
319. cout << "lab finished"<< endl;
320. return 0;
321. }