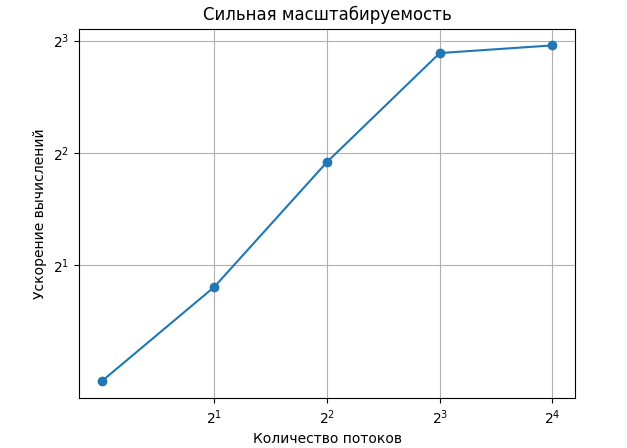
1. Постановка задачи
2. Реализовать параллельную реализацию умножения матриц DGEMM (**D**ouble precision **Ge**neral **M**atrix **M**ultiplication) из пакета BLAS с использованием OpenMP. (4 балла)
3. Привести анализ сильной/слабой масштабируемости параллельной реализации на суперкомпьютере Харизма. (2 балла)
4. Исследование на сильную масштабируемость.

M = N = K = 1024.

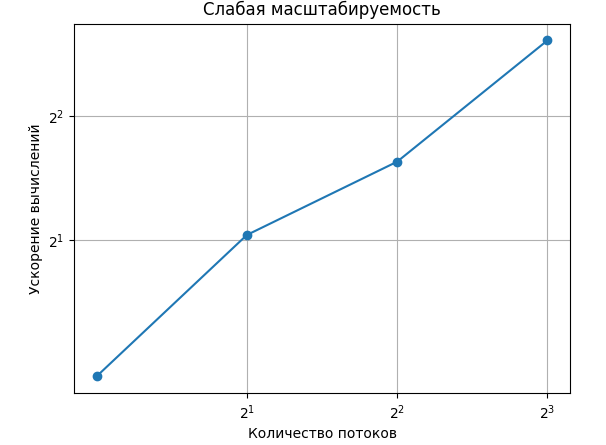
|  |  |
| --- | --- |
| Количество потоков | Время |
| 1 | 3.613285 |
| 2 | 1.9351861024 |
| 4 | 0.9252291024 |
| 8 | 0.4726181024 |
| 16 | 0.4508321024 |



1. Исследование на слабую масштабируемость.

M = N = 1024. K – меняется.

|  |  |  |  |
| --- | --- | --- | --- |
| Количество потоков | K | Время многопоточной программы | Время однопоточной программы |
| 1 | 1024 | 3.918405 | 3.491825 |
| 2 | 2048 | 4.702325 | 9.660685 |
| 4 | 4096 | 8.019382 | 24.863878 |
| 8 | 8192 | 11.052663 | 67.708891 |



1. Текст программы
2. #include <stdio.h>
3. #include <stdlib.h>
4. #include <omp.h>
5. #include <string.h>
6. #include <math.h>
7. #include <stdbool.h>
8. int flat\_id(int i, int j, int height)
9. {
10. return j \* height + i;
11. }
12. double get\_elem(int i, int j, int height, double \*M)
13. {
14. return M[flat\_id(i, j, height)];
15. }
16. void set\_elem(int i, int j, int height, double \*M, double val)
17. {
18. M[flat\_id(i, j, height)] = val;
19. }
20. char size(int height, int width)
21. {
22. return sizeof(double) \* height \* width;
23. }
24. int num\_of\_threads = 1;
25. void blas\_dgemm(int M, int N, int K, double \*A, double \*B, double \*C)
26. {
27. size\_t i, j, k;
28. #pragma omp parallel num\_threads(num\_of\_threads)
29. {
30. double sum = 0;
31. #pragma omp for
32. for (i = 0; i < M; i++)
33. {
34. for (j = 0; j < N; j++)
35. {
36. sum = 0;
37. for (k = 0; k < K; k++)
38. {
39. sum += get\_elem(i, k, M, A) \* get\_elem(k, j, K, B);
40. }
41. set\_elem(i, j, M, C, sum);
42. }
43. }
44. }
45. }
46. void printMatrix(int height, int width, double \*M)
47. {
48. for (size\_t i = 0; i < height; i++)
49. {
50. for (size\_t j = 0; j < width; j++)
51. {
52. printf("%lf ", get\_elem(i, j, height, M));
53. }
54. printf("\n");
55. }
56. }
57. void init\_matrix\_incr(int height, int width, double \*M)
58. {
59. int k = 1;
60. for (size\_t i = 0; i < height; i++)
61. {
62. for (size\_t j = 0; j < width; j++)
63. {
64. set\_elem(i, j, height, M, k);
65. k++;
66. }
67. }
68. }
69. void init\_matrix\_rand(int height, int width, double \*M)
70. {
71. for (size\_t i = 0; i < height; i++)
72. for (size\_t j = 0; j < width; j++)
73. set\_elem(i, j, height, M, (double)rand() / RAND\_MAX);
74. }
75. void init\_matrix\_val(int height, int width, double \*M, double val)
76. {
77. for (size\_t i = 0; i < height \* width; i++)
78. M[i] = val;
79. }
80. double sum\_of\_elems(int height, int width, double \*M)
81. {
82. double sum = 0;
83. for (size\_t i = 0; i < height \* width; i++)
84. sum += M[i];
85. return sum;
86. }
87. int main(int argc, char \*\*argv)
88. {
89. if (argc != 5)
90. {
91. printf("Enter arguments: \n");
92. printf("\t%s ", argv[0]);
93. printf("M ");
94. printf("N ");
95. printf("K ");
96. printf("number\_of\_threads");
97. return 1;
98. }
99. size\_t M = atoll(argv[1]);
100. size\_t K = atoll(argv[2]);
101. size\_t N = atoll(argv[3]);
102. double \*A = (double \*)malloc(M \* K \* sizeof(\*A));
103. double \*B = (double \*)malloc(K \* N \* sizeof(\*B));
104. double \*C = (double \*)malloc(M \* N \* sizeof(\*C));
105. init\_matrix\_rand(M, K, A);
106. init\_matrix\_rand(K, N, B);
107. init\_matrix\_val(M, N, C, 0.0);
108. num\_of\_threads = atoll(argv[4]);
110. printf("%d %d %d %d\n", M, N, K, num\_of\_threads);
111. size\_t repeats = 5;
112. size\_t i;
113. double time = 0;
114. for (i = 0; i < repeats; i++) {
115. double start = omp\_get\_wtime();
116. blas\_dgemm(M, N, K, A, B, C);
117. time += omp\_get\_wtime() - start;
118. }
119. time /= repeats;
120. printf("Time: %lf", time);
121. free(A);
122. free(B);
123. free(C);
124. return 0;
125. }