МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Практикум по учебному курсу

"Суперкомпьютеры и параллельная обработка данных"

**Задание №1:**

**Разработка параллельной версии программы 2 Matrix Multiplications с использованием технологии OpenMP.**

Отчет

студента 325 группы

факультета ВМК МГУ

Хайбулаева Глеба Сергеевича

2018 год

***Постановка задачи***

Распараллелить предложенную реализацию программы 2 Matrix Multiplications с использованием технологии OpenMP, а затем исследовать масштабируемость полученной программы, построить графики зависимости времени её выполнения от числа используемых ядер и объёма входных данных. Программа получает на вход 4 числа – размеры четырех матриц A, B, C и D и на основе этих данных вычисляет выражение 1.5\*A\*B\*C + 1.2\*D.

*Код программы*

1. *#include <stdio.h>*
2. *#include <unistd.h>*
3. *#include <math.h>*
4. *#include <stdlib.h>*
5. *#include <omp.h>*
6. *int NUM\_THREADS;*
7. *static*
8. *void init\_array(int ni, int nj, int nk, int nl,*
9. *double \*alpha, double \*beta,*
10. *double A [ni] [nk],*
11. *double B [nk] [nj],*
12. *double C [nj] [nl],*
13. *double D [ni] [nl])*
14. *{*
15. *int i, j;*
16. *\*alpha = 1.5;*
17. *\*beta = 1.2;*
18. *for (i = 0; i < ni; i++)*
19. *for (j = 0; j < nk; j++)*
20. *A[i][j] = (double) ((i \* j + 1) % ni) / ni;*
21. *for (i = 0; i < nk; i++)*
22. *for (j = 0; j < nj; j++)*
23. *B[i][j] = (double) (i \* (j + 1) % nj) / nj;*
24. *for (i = 0; i < nj; i++)*
25. *for (j = 0; j < nl; j++)*
26. *C[i][j] = (double) ((i \* (j + 3) + 1) % nl) / nl;*
27. *for (i = 0; i < ni; i++)*
28. *for (j = 0; j < nl; j++)*
29. *D[i][j] = (double) (i \* (j + 2) % nk) / nk;*
30. *}*
31. *static*
32. *void kernel\_2mm(int ni, int nj, int nk, int nl,*
33. *double alpha,*
34. *double beta,*
35. *double tmp[ ni][nj],*
36. *double A[ ni][nk],*
37. *double B[ nk][nj],*
38. *double C[ nj][nl],*
39. *double D[ ni][nl])*
40. *{*
41. *omp\_set\_num\_threads(NUM\_THREADS);*
42. *#pragma omp parallel for*
43. *for (int i = 0; i < ni; i++)*
44. *for (int j = 0; j < nj; j++)*
45. *{*
46. *tmp[i][j] = 0.0;*
47. *for (int k = 0; k < nk; ++k)*
48. *tmp[i][j] += alpha \* A[i][k] \* B[k][j];*
49. *}*
50. *#pragma omp barrier*
51. *omp\_set\_num\_threads(NUM\_THREADS);*
52. *#pragma omp parallel for*
53. *for (int i = 0; i < ni; i++)*
54. *for (int j = 0; j < nl; j++)*
55. *{*
56. *D[i][j] \*= beta;*
57. *for (int k = 0; k < nj; ++k)*
58. *D[i][j] += tmp[i][k] \* C[k][j];*
59. *}*
60. *}*
61. *int main(int argc, char\*\* argv)*
62. *{*
63. *double alpha;*
64. *double beta;*
65. *int nums[8] = {1, 2, 4, 8, 16, 32, 64, 128};*
66. *int nis[5] = {16, 40, 180, 800, 1600};*
67. *int njs[5] = {18, 50, 190, 900, 1800};*
68. *int nks[5] = {22, 70, 210, 1100, 2200};*
69. *int nls[5] = {24, 80, 220, 1200, 2400};*
70. *for (int t = 0; t < 5; t++) {*
71. *int ni = nis[t];*
72. *int nk = nks[t];*
73. *int nj = njs[t];*
74. *int nl = nls[t];*
75. *printf("ni = %d, nk = %d, nj = %d, nl = %d\n \n", ni, nk, nj, nl);*
76. *double (\*tmp)[ni][nj]; tmp = (double(\*)[ni][nj])malloc ((ni) \* (nj) \* sizeof(double));*
77. *double (\*A)[ni][nk]; A = (double(\*)[ni][nk])malloc ((ni) \* (nk) \* sizeof(double));*
78. *double (\*B)[nk][nj]; B = (double(\*)[nk][nj])malloc ((nk) \* (nj) \* sizeof(double));*
79. *double (\*C)[nj][nl]; C = (double(\*)[nj][nl])malloc ((nj) \* (nl) \* sizeof(double));*
80. *double (\*D)[ni][nl]; D = (double(\*)[ni][nl])malloc ((ni) \* (nl) \* sizeof(double));*
81. *init\_array (ni, nj, nk, nl, &alpha, &beta,*
82. *\*A,*
83. *\*B,*
84. *\*C,*
85. *\*D);*
86. *for (int i = 0; i < 8; i++) {*
87. *NUM\_THREADS = nums[i];*
88. *double start = omp\_get\_wtime();*
89. *kernel\_2mm (ni, nj, nk, nl,*
90. *alpha, beta,*
91. *\*tmp,*
92. *\*A,*
93. *\*B,*
94. *\*C,*
95. *\*D);*
96. *double end = omp\_get\_wtime();*
97. *printf("Number of threads = %d\n", NUM\_THREADS);*
98. *printf("Time = %.6f", end - start);*
99. *printf("\n");*
100. *}*
101. *free((void\*)tmp);*
102. *free((void\*)A);*
103. *free((void\*)B);*
104. *free((void\*)C);*
105. *free((void\*)D);*
106. *printf("\n");*
107. *}*
108. *return 0;*
109. *}*

Распараллелена программа классическими приёмами, рассмотренными на лекции.

***Результаты замеров времени выполнения***

Работа задачи рассмотрена на суперкомпьютере Polus с различным числом нитей (1 - 128) и различными наборами данных, предоставленных вместе с исходным кодом (см ниже). Каждое измерение проводилось 3 раза. В таблице и на графиках записаны усредненные результаты времени выполнения.  
Наборы данных:  
Mini – 16, 18, 22, 24  
Small – 40, 50, 70, 80  
Medium – 180, 190, 210, 220  
Large – 800, 900, 1100, 1200  
Extralarge – 1600, 1800, 2200, 2400

***Таблица с результатами***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Нити/размеры матриц | Mini | Small | Medium | Large | Extralarge |
| 1 | 0.015268 | 0.001817 | 0.085185 | 41.779608 | 92.493467 |
| 2 | 0.029977 | 0.000904 | 0.043623 | 5.649542 | 47.945681 |
| 4 | 0.000169 | 0.000465 | 0.022020 | 2.735370 | 24.690036 |
| 8 | 0.000214 | 0.000281 | 0.011592 | 1.394740 | 15.057165 |
| 16 | 0.000366 | 0.000230 | 0.006024 | 0.723245 | 8.306721 |
| 32 | 0.000904 | 0.255991 | 0.003105 | 0.404365 | 4.497701 |
| 64 | 0.058304 | 0.059921 | 0.001371 | 0.513174 | 3.021916 |
| 128 | 0.460011 | 0.160068 | 0.135570 | 0.446294 | 2.797628 |

***Графики: время выполнения программы в зависимости от размеров матриц и количества потоков***

В виде линий:

В виде поверхности:

В виде линий в трехмерном пространстве:

***Вывод:***

Распараллеливание программы в среднем дало выигрыш по времени в 4 раза на массивах большого размера.

Как показали эксперименты, для малых рассмотренных наборах данных (Mini, Small) выгодно использовать 4-16 нитей, а для больших (Medium, Large, Extralarge) – 32-128. При увеличении числа нитей у наборов данных малого размера наблюдается снижение производительности, что происходит из-за больших накладных расходов, в то время как производительность на больших наборах возрастает с увеличением числа нитей, т.к. в этом случае накладные расходы не столь существенны по сравнению с общими затрачиваемыми ресурсами.

Отсюда следует вывод, что на больших наборах данных стоит использовать большее число нитей, т.к. при этом наблюдается существенный выигрыш по времени выполнения программы при распараллеливании с помощью технологии OpenMP, но на маленьких стоит использовать небольшое количество нитей из-за существенных накладных расходов при использовании большого числа нитей.